

```

BBBBBBBBBBBBBB      AAAAAAAAAA      SSSSSSSSSSSSSS      RRRRRRRRRRRR      TTTTTTTTTTTTTTTT      LLL
BBBBBBBBBBBBBB      AAAAAAAAAA      SSSSSSSSSSSSSS      RRRRRRRRRRRR      TTTTTTTTTTTTTTTT      LLL
BBBBBBBBBBBBBB      AAAAAAAAAA      SSSSSSSSSSSSSS      RRRRRRRRRRRR      TTTTTTTTTTTTTTTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAAAAAAAAAAAAAAAAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAAAAAAAAAAAAAAAAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAAAAAAAAAAAAAAAAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBB      BBB      AAA      AAA      SSS      SSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSSSSSSSSSSSSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSSSSSSSSSSSSS      RRR      RRR      TTT      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSSSSSSSSSSSSS      RRR      RRR      TTT      TTT      LLL

```

```
BBBBBBBBB  AAAAAA  SSSSSSSS  EEEEEEEEE  RRRRRRRR  RRRRRRRR  000000  RRRRRRRR
BBBBBBBBB  AAAAAA  SSSSSSSS  EEEEEEEEE  RRRRRRRR  RRRRRRRR  000000  RRRRRRRR
BB      BB  AA      AA  SS      SS  EE      EE  RR      RR  RR      RR  00      00  RR      RR
BB      BB  AA      AA  SS      SS  EE      EE  RR      RR  RR      RR  00      00  RR      RR
BB      BB  AA      AA  SS      SS  EE      EE  RR      RR  RR      RR  00      00  RR      RR
BB      BB  AA      AA  SS      SS  EE      EE  RR      RR  RR      RR  00      00  RR      RR
BBBBBBBBB  AA      AA  SSSSSS  EEEEEEE  RRRRRRRR  RRRRRRRR  00      00  RRRRRRRR
BBBBBBBBB  AA      AA  SSSSSS  EEEEEEE  RRRRRRRR  RRRRRRRR  00      00  RRRRRRRR
BB      BB  AAAAAAAAAA  SS      SS  EE      EE  RR      RR  RR      RR  00      00  RR      RR
BB      BB  AAAAAAAAAA  SS      SS  EE      EE  RR      RR  RR      RR  00      00  RR      RR
BB      BB  AA      AA  SS      SS  EE      EE  RR      RR  RR      RR  00      00  RR      RR
BB      BB  AA      AA  SSSSSSSS  EEEEEEEEE  RR      RR  RR      RR  00      00  RR      RR
BBBBBBBBB  AA      AA  SSSSSSSS  EEEEEEEEE  RR      RR  RR      RR  000000  RR      RR
BBBBBBBBB  AA      AA  SSSSSSSS  EEEEEEEEE  RR      RR  RR      RR  000000  RR      RR
                                     ....
                                     ....
                                     ....
                                     ....
```

```
LL      LL      SSSSSSSS
LL      LL      SSSSSSSS
LL      LL      SS
LL      LL      SS
LL      LL      SS
LL      LL      SS
LL      LL      SSSSSS
LL      LL      SSSSSS
LL      LL      SS
LL      LL      SS
LL      LL      SS
LL      LL      SS
LLLLLLLLLL  IIIIIII  SSSSSSSS
LLLLLLLLLL  IIIIIII  SSSSSSSS
```

```
0001 0 MODULE BASERROR (
0002 0 IDENT = '1-074'
0003 0 ) =
0004 1 BEGIN
0005 1
0006 1 *****
0007 1 *
0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0010 1 * ALL RIGHTS RESERVED.
0011 1 *
0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0017 1 * TRANSFERRED.
0018 1 *
0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0021 1 * CORPORATION.
0022 1 *
0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0025 1 *
0026 1 *****
0027 1
0028 1
0029 1
0030 1
0031 1 **
0032 1 FACILITY: VAX-11 BASIC Error Handling
0033 1
0034 1 ABSTRACT:
0035 1
0036 1 This module contains the VAX-11 BASIC error handling logic.
0037 1 The error data base is OWN to this module.
0038 1
0039 1 ENVIRONMENT: VAX-11 user mode
0040 1
0041 1 AUTHOR: John Sauter, CREATION DATE: 17-Oct-78
0042 1
0043 1 MODIFIED BY:
0044 1
0045 1 1-001 - Original. JBS 27-NOV-78
0046 1 1-002 - Remove BAS$$SIGNAL_IO and BAS$$STOP_IO. They now live in
0047 1 their own module. JBS 08-DEC-78
0048 1 1-003 - Add global definitions of BAS$_abcxyz symbols. JBS 11-DEC-78
0049 1 1-004 - Include severity in those definitions. JBS 19-DEC-78
0050 1 1-005 - If the compiled code does not do any error processing,
0051 1 either continue, restart the line or exit. JBS 28-DEC-78
0052 1 1-006 - Call BAS$$CB_CLEANUP to flush active I/O when unwinding.
0053 1 JBS 29-DEC-78
0054 1 1-007 - Change BAS$$CB_CLEANUP to OT$$CLEANUP_IO. JBS 09-JAN-1979
0055 1 1-008 - When restarting an I/O statement, do an UNWIND to the
0056 1 beginning of the I/O statement. JBS 26-JAN-1979
0057 1 1-009 - Remove OT$$CLEANUP_IO, since we will do I/O cleanup using
```


58	0058	1	a stack frame. JBS 26-JAN-1979
59	0059	1	1-010 - When searching for a line number corresponding to a PC
60	0060	1	look in the right place in the table. JBS 30-JAN-1979
61	0061	1	1-011 - When getting storage for the SIGNAL argument list, get enough
62	0062	1	for the argument count and the two trailing longwords, even
63	0063	1	though this may sometimes be a little more than is needed.
64	0064	1	JBS 31-JAN-1979
65	0065	1	1-012 - Purge the terminal output buffer before printing an error
66	0066	1	message. JBS 02-FEB-1979
67	0067	1	1-013 - Add support for I/O lists and change the name of the prefix
68	0068	1	for stack frames from BAS\$ to BSF\$. JBS 08-FEB-1979
69	0069	1	1-014 - Because control C puts some non-BASIC frames on the stack,
70	0070	1	be cleverer about searching through stack frames for a
71	0071	1	non-GOSUB frame. JBS 20-FEB-1979
72	0072	1	1-015 - In BAS\$\$SIGNAL, don't force the severity to SEVERE ERROR
73	0073	1	by calling LIB\$STOP. JBS 20-FEB-1979
74	0074	1	1-016 - Search the PC table from back to front so that the line numbers
75	0075	1	from statements which generate no code, such as DATA statements,
76	0076	1	will not appear. JBS 22-FEB-1979
77	0077	1	1-017 - Use OT\$\$PUR_10_ERR to purge I/O buffers, thus avoiding having
78	0078	1	to REQUIRE all of the I/O data structures. JBS 07-MAR-1979
79	0079	1	1-018 - Concatenate a ?, X or space on the front of error messages
80	0080	1	in BAS\$ERR based on the severity of the error. JBS 12-MAR-1979
81	0081	1	1-019 - In BAS\$ERR, don't clobber the length field of a dynamic
82	0082	1	string. JBS 22-MAR-1979
83	0083	1	1-020 - Change name of ILLEGAL RESUME. JBS 02-APR-1979
84	0084	1	1-021 - Make BAS\$\$COND VAL global, so BAS\$\$SIGNAL_10 can use it.
85	0085	1	JBS 06-APR-1979
86	0086	1	1-022 - Only restart statements after restartable I/O failures if
87	0087	1	the I/O was to a terminal. JBS 06-APR-1979
88	0088	1	1-023 - RESUME with no line number will resume into another module.
89	0089	1	JBS 12-APR-1979
90	0090	1	1-024 - The compiled code can get S\$\$SUBRNG. JBS 15-APR-1979
91	0091	1	1-025 - Correct an error in edit 022. JBS 16-APR-1979
92	0092	1	1-026 - Correct an error in unwinding from a RESUME with no
93	0093	1	line number. JBS 30-APR-1979
94	0094	1	1-027 - If the line number is not found, take the line number
95	0095	1	corresponding to the next earlier PC. This is needed
96	0096	1	because (contrary to the specification) the compiler
97	0097	1	does not put its "fake line numbers" in the line number
98	0098	1	table. JBS 04-MAY-1979
99	0099	1	1-028 - If we are restarting an I/O statement, call BAS\$\$RESTART_10
100	0100	1	to reinitialize the I/O data base. JBS 07-MAY-1979
101	0101	1	1-029 - If we are doing system handling on an INFO message, don't
102	0102	1	promote it to a warning. JBS 10-MAY-1979
103	0103	1	1-030 - If we convert a system message to a BASIC message, be sure
104	0104	1	the PC and PSL of the failure are reported. JBS 11-MAY-1979
105	0105	1	1-031 - Publish the PC and PSL for any converted message.
106	0106	1	JBS 13-MAY-1979
107	0107	1	1-032 - Include certain string error codes in the list of messages
108	0108	1	which are converted to BASIC-specific errors. JBS 16-MAY-1979
109	0109	1	1-033 - Convert LIB\$\$ and OT\$\$ to STR\$. JBS 21-MAY-1979
110	0110	1	1-034 - Correct an error in BAS\$\$USER HAND which prevented intercepting
111	0111	1	an error that had once been through ON ERROR GO BACK.
112	0112	1	JBS 29-MAY-1979
113	0113	1	1-035 - Add BAS\$\$ERR INIT. JBS 04-JUN-1979
114	0114	1	1-036 - Call BAS\$\$UNWIND when cutting back a frame. JBS 06-JUN-1979

115	0115	1	1-037 - Defer calling SYSSUNWIND to the top level handler.
116	0116	1	JBS 06-JUN-1979
117	0117	1	1-038 - BAS\$ERR INIT must clear SYSTEM_ERROR and GONE_BACK.
118	0118	1	JBS 07-JUN-1979
119	0119	1	1-039 - RESUME to a line number must accumulate the number of frames
120	0120	1	to unwind. JBS 10-JUL-1979
121	0121	1	1-040 - Change call to STR\$COPY. JBS 16-JUL-1979
122	0122	1	1-041 - Fix a bug which caused GONE_BACK to remain set after an UNWIND.
123	0123	1	JBS 23-JUL-1979
124	0124	1	1-042 - When unwinding to a frame, POP its I/O. JBS 24-JUL-1979
125	0125	1	1-043 - Change call to OT\$ST\$TERM IO. JBS 26-JUL-1979
126	0126	1	1-044 - Remove edit 023: don't allow RESUME into another module.
127	0127	1	JBS 26-JUL-1979
128	0128	1	1-045 - Give error 31 (illegal byte count for I/O) in response to
129	0129	1	an attempt to do I/O to a closed file. JBS 01-AUG-1979
130	0130	1	1-046 - Don't try to build an argument list for LIB\$SIGNAL longer
131	0131	1	than 255. JBS 08-AUG-1979
132	0132	1	1-047 - Correct a typo in edit 044. JBS 20-AUG-1979
133	0133	1	1-048 - Call BAS\$SPUR IO ERR. JBS 20-AUG-1979
134	0134	1	1-049 - Translate MTH\$FCOOVEMAT into floating overflow, since it
135	0135	1	is produced by both the EXP and TAN functions. JBS 20-AUG-1979
136	0136	1	1-050 - Change BAS\$HANDLER to BAS\$\$HANDLER for the sharable library.
137	0137	1	JBS 20-AUG-1979
138	0138	1	1-051 - Move the definitions of the error codes to BAS\$MSGDEF, for
139	0139	1	the sake of the shared library. JBS 21-AUG-1979
140	0140	1	1-052 - Remove the redundant RETURN statement, the BLISS compiler no
141	0141	1	longer needs it. JBS 06-SEP-1979
142	0142	1	1-053 - Add BAS\$PUSH_ERR and BAS\$POP_ERR. JBS 10-SEP-1979
143	0143	1	1-054 - Change IOL from I/O list to Immediate On-Line. JBS 10-SEP-1979
144	0144	1	1-055 - If a BASIC condition is signalled as INFO, don't promote
145	0145	1	it to a more severe condition. This is needed for the
146	0146	1	two kinds of control C signals for the RUN command.
147	0147	1	JBS 14-SEP-1979
148	0148	1	1-056 - Change MTH\$SINCOSSIG to MTH\$SIGLOSMAT. JBS 19-SEP-1979
149	0149	1	1-057 - Add STR\$STRTOOLON. JBS 31-OCT-1979
150	0150	1	1-058 - Make ERR, ERL and ERN\$ retain their values after RESUME.
151	0151	1	JBS 07-NOV-1979
152	0152	1	1-059 - Fix restarting an I/O statement to clear the error flag.
153	0153	1	JBS 08-NOV-1979
154	0154	1	1-060 - Make sure that a user error handler doesn't try to handle
155	0155	1	INFO conditions. This is a part of edit 055. JBS 15-NOV-1979
156	0156	1	1-061 - Handle correctly a main program with ON ERROR GO BACK getting
157	0157	1	a restartable error. JBS 09-JAN-1980
158	0158	1	1-062 - Handle delta PC values greater than 2^15. JBS 12-FEB-1980
159	0159	1	1-063 - Handle error trapping in a module without line numbers, except for
160	0160	1	RESUME with no line number. JBS 07-MAR-1980
161	0161	1	1-064 - Treat floating faults the same as traps IN BAS\$\$HANDLER. SBL 10-Jun-1980
162	0162	1	1-065 - Distinguish between a major and a minor frame in BAS\$\$USER HAND, so
163	0163	1	when an error is ON ERROR GO BACK: 0 in a minor frame the major
164	0164	1	frame can handle the error. FM 13-FEB-81.
165	0165	1	1-066 - Comments referring to SYSMSG.MPF are using an obsolete name; the name
166	0166	1	should be SYSSMESSAGE:SYSMSG.EXE. PL 26-Aug-81
167	0167	1	1-067 - Convert SSS_DECOVF to equivalent BAS\$ error. PLL 5-Apr-1982
168	0168	1	1-068 - Remove code that was a workaround for a bug in SYSSUNWIND (could not
169	0169	1	be called with an argument of zero). In BAS\$\$USER HAND, instead of
170	0170	1	patching the return PC of the frame that returns to the compiled code,
171	0171	1	do nothing and let BAS\$\$HANDLER call SYSSUNWIND. This fixes a bug


```

: 172      0172 1  with integer overflow that occurs when the g, h emulator is invoked.
: 173      0173 1  PLL 26-Apr-82
: 174      0174 1  1-069 - Back out the last edit. It broke STOP and possibly some other
: 175      0175 1  things. Apparently calling SYSSUNWIND with zero does not always
: 176      0176 1  work as expected. So put in a more obscure fix (in BAS$$HANDLER)
: 177      0177 1  for integer overflow with the g, h emulator. PLL 11-May-1982
: 178      0178 1  1-070 - BAS$$HANDLER must check to see how many arguments in the signal
: 179      0179 1  argument list - sometimes there is no PSW. PLL 11-May-1982
: 180      0180 1  1-071 - Give the new error message 'improper error handling' if a main
: 181      0181 1  program calls a subprogram in its error handler, and this sub-
: 182      0182 1  program does an ON ERROR GO BACK. PLL 22-Jun-1982
: 183      0183 1  1-072 - make variables BAS$T_ERN, BAS$L_ERR and BAS$L_ERL globals, so
: 184      0184 1  that they are accessible from other routines (specifically BAS$CTRLC)
: 185      0185 1  also make BAS$$LINE & BAS$$MODULE global routines. MDL 22-Jul-1982
: 186      0186 1  1-073 - BAS$$HANDLER should be prepared to restart an ANSI INPUT error.
: 187      0187 1  PLL 29-Jul-1982
: 188      0188 1  1-074 - don't set BAS$L GOING BACK in BAS$$USER_HAND if this is a restartable
: 189      0189 1  error. MDL 29-Mar-1984
: 190      0190 1  !--
: 191      0191 1
: 192      0192 1  !<BLF/PAGE>

```

```
194 0193 1 |
195 0194 1 | SWITCHES:
196 0195 1 |
197 0196 1 |
198 0197 1 SWITCHES ADDRESSING_MODE (EXTERNAL = GENERAL, NONEXTERNAL = WORD_RELATIVE);
199 0198 1 |
200 0199 1 |
201 0200 1 | LINKAGES:
202 0201 1 |
203 0202 1 |
204 0203 1 LINKAGE
205 0204 1     RESTART_LINK = JSB : NOTUSED (11, 10, 9)
206 0205 1     NOPRESERVE (8, 7, 6, 5, 4, 3, 2);
207 0206 1 |
208 0207 1 REQUIRE 'RTLIN:OTSLNK';
209 0636 1 | Define common linkages
210 0637 1 |
211 0638 1 | TABLE OF CONTENTS:
212 0639 1 |
213 0640 1 |
214 0641 1 FORWARD ROUTINE
215 0642 1     BAS$$SIGNAL : NOVALUE,
216 0643 1     BAS$$STOP : NOVALUE,
217 0644 1     BAS$$COND_VAL,
218 0645 1     PC_TO_LINE_NO,
219 0646 1     BAS$$LINE,
220 0647 1     BAS$$FUNCTION,
221 0648 1     BAS$$MODULE,
222 0649 1     HANDLER_HANDLER,
223 0650 1     RESTART : RESTART_LINK NOVALUE,
224 0651 1     RESTART_IO : RESTART_LINK NOVALUE,
225 0652 1     BAS$$USER_HAND,
226 0653 1     BAS$$RESUME,
227 0654 1     BAS$$RESUME_Z,
228 0655 1     BAS$$ON_ERR_Z,
229 0656 1     BAS$$ON_ERR_BK,
230 0657 1     BAS$$HANDLER,
231 0658 1     BAS$ERR,
232 0659 1     BAS$ERR,
233 0660 1     BAS$ERR,
234 0661 1     BAS$ERR,
235 0662 1     BAS$ERROR : NOVALUE,
236 0663 1     BAS$$ERR_INIT : NOVALUE,
237 0664 1     BAS$PUSH_ERR,
238 0665 1     BAS$POP_ERR;
239 0666 1 |
240 0667 1 |
241 0668 1 | INCLUDE FILES:
242 0669 1 |
243 0670 1 |
244 0671 1 LIBRARY 'RTLSTARLE';
245 0672 1 |
246 0673 1 REQUIRE 'RTLIN:RTLPSECT';
247 0768 1 |
248 0769 1 REQUIRE 'RTLIN:BASFRAME';
249 0972 1 |
250 0973 1 REQUIRE 'RTLIN:BASERRMSG';
```

```
| signal an error
| signal an error
| compute VAX/VMS cond value
| convert PC to line number
| get the number of the current line
| get the name of the current function
| get the name of the current module
| handler for BAS$HANDLER
| unwind target
| unwind target for I/O
| try to let user handle error
| resume from a condition handler
| likewise, but no line number
| ON ERROR GOTO 0
| ON ERROR GO BACK
| handle a BASIC-PLUS-2 error
| return error line number
| return current error number
| return module name of error
| return text of error number
| signal an error from compiled code
| Initialize for the RUN command
| Save error info
| Restore error info
```

```
| system symbols
| macros to declare psects
| define frame structure
| Define ERROR_LIST macro.
```

```
251 1596 1
252 1597 1 REQUIRE 'RTLML:OTSLUB';           ! Define LUB
253 1737 1
254 1738 1
255 1739 1 MACROS:
256 1740 1
257 1741 1     NONE
258 1742 1
259 1743 1 EQUATED SYMBOLS:
260 1744 1
261 1745 1 +
262 1746 1 Define the special error codes used for I/O errors and traceback.
263 1747 1 -
264 1748 1
265 1749 1 LITERAL
266 1750 1     ERR_TRACE_MAIN = 4089,           ! main program
267 1751 1     ERR_TRACE_SUB = 4090,         ! external subroutine
268 1752 1     ERR_TRACE_EXTF = 4091,        ! external function
269 1753 1     ERR_TRACE_DEF = 4092,         ! DEF procedure
270 1754 1     ERR_TRACE_DEFS = 4093,        ! DEF* procedure
271 1755 1     ERR_TRACE_GOSB = 4094,        ! GOSUB
272 1756 1     ERR_TRACE_ONER = 4095,        ! condition handler
273 1757 1     ERR_TRACE_IOLST = 4087,       ! immediate mode code
274 1758 1     ERR_TRACE_PCPSL = 4086;       ! user PC=!XL, PSL=!XL
275 1759 1
276 1760 1 +
277 1761 1 Define the return values from BAS$$USER_HAND.
278 1762 1 -
279 1763 1
280 1764 1 LITERAL
281 1765 1     USER_HAND_CONT = 0,             ! Continue from point of error (or of unwind)
282 1766 1     USER_HAND_BACK = 1,            ! Try caller's handler
283 1767 1     USER_HAND_FAIL = 2;           ! Force system error processing
284 1768 1
285 1769 1 +
286 1770 1 Define the return values from the user's error handler.
287 1771 1 None of those below implies RESUME with a line number.
288 1772 1 -
289 1773 1
290 1774 1 LITERAL
291 1775 1     USER_ERR_RSUMZ = 0,               ! RESUME with no line number
292 1776 1     USER_ERR_GOBK = 1,              ! ON ERROR GO BACK
293 1777 1     USER_ERR_OEGZ = 2;             ! ON ERROR GOTO 0
294 1778 1
295 1779 1 +
296 1780 1 Define the coded values for system error handling.
297 1781 1 -
298 1782 1
299 1783 1 LITERAL
300 1784 1     K_SYS_CONT = 1,                   ! Continue in line
301 1785 1     K_SYS_EXIT = 2,                  ! Exit the image (LIB$STOP)
302 1786 1     K_SYS_RESTART = 3;               ! Restart the line which had the error
303 1787 1
304 1788 1
305 1789 1 PSECTS:
306 1790 1
307 1791 1 DECLARE_PSECTS (BAS);
```



```
308 1792 1 |
309 1793 1 |   OWN STORAGE:
310 1794 1 |
311 1795 1 | GLOBAL
312 1796 1 |   BAS$T_ERN : BLOCK [8, BYTE] INITIAL (BYTE ( REP 8 OF (0))), ! descriptor for module name
313 1797 1 |   BAS$L_ERR : INITIAL (0), ! current error code
314 1798 1 |   BAS$L_ERL : INITIAL (0); ! line number of error
315 1799 1 |
316 1800 1 |   OWN
317 1801 1 |   BAS$L_ERRFLG : INITIAL (0), ! 1 = error in progress
318 1802 1 |   HIGHEST_LEVEL : INITIAL (0), ! Level to unwind to on RESUME
319 1803 1 |   HIGHEST_FMP : INITIAL (0), ! Frame to unwind to on RESUME
320 1804 1 |   ACCUM_LEVEL : INITIAL (0), ! Level to unwind to on RESUME with a line number
321 1805 1 |   UNWIND_COUNT : INITIAL (0), ! Level for top handler to unwind
322 1806 1 |   SYSTEM_ERROR : INITIAL (0), ! Set for "fatal fatal" error
323 1807 1 |   GONE_BACK : INITIAL (0), ! Set for ON ERROR GO BACK
324 1808 1 |   ERROR_STACK : VOLATILE VECTOR [2] INITIAL (0, 0), ! Error stack
325 1809 1 |   ERROR_STACK_INI : VOLATILE INITIAL (0); ! Init flag for ERROR_STACK
326 1810 1 |
327 1811 1 | +
328 1812 1 | Some OWN storage is needed so that communication can take place
329 1813 1 | between levels of BAS$$USER_HAND and to RESTART.
330 1814 1 | -
331 1815 1 |
332 1816 1 |   OWN
333 1817 1 |   BAS$A_CH_CUR_LN : INITIAL (0), ! restart PC
334 1818 1 |   BAS$L_GOING_BACK : INITIAL (0), ! 1 when "going back"
335 1819 1 |   BAS$A_RESTART : INITIAL (0); ! restart PC
336 1820 1 |
337 1821 1 |
338 1822 1 |   EXTERNAL REFERENCES:
339 1823 1 |
340 1824 1 |
341 1825 1 |   EXTERNAL ROUTINE
342 1826 1 |   LIB$MATCH_COND, ! match condition codes
343 1827 1 |   LIB$SIGNAL, ! system error signaller
344 1828 1 |   LIB$STOP, ! system fatal error signaller
345 1829 1 |   SY$$UNWIND, ! unwind the stack
346 1830 1 |   LIB$FIXUP_FLT, ! fix up reserved operands
347 1831 1 |   LIB$GET_VM, ! get storage
348 1832 1 |   LIB$FREE_VM, ! free storage
349 1833 1 |   STR$CONCAT, ! Concatenate two strings
350 1834 1 |   STR$COPY_R, ! Copy a string by ref
351 1835 1 |   STR$COPY_DX, ! Copy a string by desc
352 1836 1 |   SY$$GETMSG, ! get the message text for a signal condition
353 1837 1 |   BAS$INIT_ONERR, ! run a condition handler
354 1838 1 |   BAS$$RESTART_IO, ! Restart an I/O statement
355 1839 1 |   BAS$$PUR_IO_ERR : NOVALUE, ! Purge I/O on an error
356 1840 1 |   OT$$TERM_IO, ! Test for terminal I/O
357 1841 1 |   BAS$$UNWIND : NOVALUE, ! Purge a frame
358 1842 1 |   BAS$$UNWIND_IO : NOVALUE, ! Purge a frame's I/O
359 1843 1 |   BAS$HANDLER; ! Header for condition handler
360 1844 1 |
361 1845 1 | +
362 1846 1 | The following symbols are defined in module BAS$MSGDEF
363 1847 1 | -
364 1848 1 |
```

```
365 1849 1 EXTERNAL LITERAL
366 1850 1 BASSK_FAC NO : UNSIGNED (12)
367 1851 1 BASSK_RESNO ERR : UNSIGNED (8),
368 1852 1 BASSK_ILLRESSUB : UNSIGNED (8),
369 1853 1 BASSK_DIVBY ZER : UNSIGNED (8),
370 1854 1 BASSK_IMASQOORO : UNSIGNED (8),
371 1855 1 BASSK_ILLARGLOG : UNSIGNED (8),
372 1856 1 BASSK_INTERR : UNSIGNED (8),
373 1857 1 BASSK_MEMMANVIO : UNSIGNED (8),
374 1858 1 BASSK_FLOPOIERR : UNSIGNED (8),
375 1859 1 BASSK_SUBOUTRAN : UNSIGNED (8),
376 1860 1 BASSK_MAXMEMEXC : UNSIGNED (8),
377 1861 1 BASSK_ILLBYTCOU : UNSIGNED (8),
378 1862 1 BASS_ON CHAFIL,
379 1863 1 BASSK_PROLOSSOR : UNSIGNED (8),
380 1864 1 BASSK_STRTOOLON : UNSIGNED (8),
381 1865 1 BASSK_DECERR : UNSIGNED (8),
382 1866 1 BASSK_IMPERRHAN : UNSIGNED (8);
383 1867 1
384 1868 1 The following VAX/VMS condition codes are used in this module
385 1869 1
386 1870 1
387 1871 1 EXTERNAL LITERAL
388 1872 1
389 1873 1 Attempt to compute the square root of a negative number.
390 1874 1 The result will be the reserved operand.
391 1875 1
392 1876 1 MTHS_SQUROONEG,
393 1877 1
394 1878 1 Attempt to compute the logarithm of 0, or of a negative number.
395 1879 1 The result will be the reserved operand.
396 1880 1
397 1881 1 MTHS_LOGZERNEG,
398 1882 1
399 1883 1 Attempt to raise E to a power so large that the result cannot
400 1884 1 be represented by the computer. That power is about 88. The
401 1885 1 result will be the reserved operand.
402 1886 1
403 1887 1 MTHS_FLOOVEMAT,
404 1888 1
405 1889 1 Attempt to raise a base, B, to a power, P, where this is undefined.
406 1890 1 For example, 0 raised to the 0 power is undefined. The result
407 1891 1 will be the reserved operand.
408 1892 1
409 1893 1 MTHS_UNDEXP,
410 1894 1
411 1895 1 Attempt to take the SINE or COSINE of a number so large that,
412 1896 1 after taking the number modulo 2 PI, there is no information
413 1897 1 left. This is caused by the fact that the computer keeps
414 1898 1 only the highest-order significant bits of a number. The
415 1899 1 result will be the reserved operand. Other functions in the
416 1900 1 math library may also signal this under similar conditions.
417 1901 1
418 1902 1 MTHS_SIGLOSMAT,
419 1903 1
420 1904 1 Attempt to allocate a dynamic string when there is
421 1905 1 not enough virtual memory left to hold it along with
```



```
422 1906 1 | all of the other strings allocated.
423 1907 1 |
424 1908 1 |   STR$_INSVIRMEM,
425 1909 1 |
426 1910 1 | Divide by zero in string arithmetic.
427 1911 1 |
428 1912 1 |   STR$_DIVBY_ZER,
429 1913 1 |
430 1914 1 | Attempt to create a string longer than 65535 characters,
431 1915 1 | the maximum length allowed by the VAX-11 string architecture.
432 1916 1 | This can be the result of, for example, the concatenation of
433 1917 1 | two 50,000 character strings.
434 1918 1 |
435 1919 1 |   STR$_STRTOOLON,
436 1920 1 |
437 1921 1 | Attempt to continue to do I/O to a closed file.
438 1922 1 | (That is, the file was closed between element
439 1923 1 | transmitters, and another element transmission
440 1924 1 | was attempted.)
441 1925 1 |
442 1926 1 |   OTS$_IO_CONCLO;
443 1927 1 |
444 1928 1 | Attempt to compute a packed decimal result which the computer
445 1929 1 | can not represent.
446 1930 1 |
447 1931 1 |   SSS$_DECOVF           ! (defined in RTLSTARLE)
448 1932 1 |
449 1933 1 |
450 1934 1 | Attempt to divide a real number by 0.
451 1935 1 |
452 1936 1 |   SSS$_FLTDIV           ! (defined in RTLSTARLE)
453 1937 1 |   SSS$_FLTDIV_F (fault) ! (defined in RTLSTARLE)
454 1938 1 |
455 1939 1 | Attempt to divide an integer by 0.
456 1940 1 |
457 1941 1 |   SSS$_INTDIV           ! (defined in RTLSTARLE)
458 1942 1 |
459 1943 1 | Attempt to compute a floating point result which the computer
460 1944 1 | cannot represent.
461 1945 1 |
462 1946 1 |   SSS$_FLTTOVF           ! (defined in RTLSTARLE)
463 1947 1 |   SSS$_FLTTOVF_F (fault) ! (defined in RTLSTARLE)
464 1948 1 |
465 1949 1 | Attempt to compute an integer result which the computer cannot
466 1950 1 | represent.
467 1951 1 |
468 1952 1 |   SSS$_INTTOVF           ! (defined in RTLSTARLE)
469 1953 1 |
470 1954 1 | Reserved operand fault. In the context of BASIC, this is usually
471 1955 1 | caused by an attempt to refer to a reserved floating operand, but
472 1956 1 | it can be caused by other errors. Only the floating reserved
473 1957 1 | operand case is handled by BASIC.
474 1958 1 |
475 1959 1 |   SSS$_ROPRAND           ! (define in RTLSTARLE)
476 1960 1 |
477 1961 1 | Attempt to refer to an invalid address. This can happen if
478 1962 1 | range checking on array indicies is defeated.
```

```

: 479      1963 1 |
: 480      1964 1 |      SSS_ACCVIO          ! (defined in RTLSTARLE)
: 481      1965 1 |
: 482      1966 1 |      Attempt to use an index outside its proper range. This can happen
: 483      1967 1 |      when the compiler generates in-line array indexing.
: 484      1968 1 |
: 485      1969 1 |      SSS_SUBRNG          ! (defined in RTLSTARLE)
: 486      1970 1 |
: 487      1971 1 |
: 488      1972 1 |      EXTERNAL
: 489      1973 1 |          OTSS$A_CUR_LUB : ADDRESSING MODE (GENERAL);
: 490      1974 1 |          ! Addr of current LUB/ISB/RAB
: 491      1975 1 |      !<BLF/PAGE>

```



```
493 1976 1 |
494 1977 1 | Define the severity code for each possible error.
495 1978 1 |
496 1979 1 |
497 1980 1 | MACRO
498 M 1981 1 |   ERR (NUMBER, CODE, TEXT, SEVERITY, SYSTEM_HANDLING) =
499 M 1982 1 |     (XIF (XIDENTICAL (SEVERITY, WARNING)) XTHEN ST$K WARNING XELSE
500 M 1983 1 |       XIF (XIDENTICAL (SEVERITY, SUCCESS)) XTHEN ST$K SUCCESS XELSE
501 M 1984 1 |       XIF (XIDENTICAL (SEVERITY, ERROR)) XTHEN ST$K ERROR XELSE
502 M 1985 1 |       XIF (XIDENTICAL (SEVERITY, INFO)) XTHEN ST$K INFO XELSE
503 M 1986 1 |       XIF (XIDENTICAL (SEVERITY, SEVERE)) XTHEN ST$K_SEVERE XELSE
504 M 1987 1 |       257 XFI XFI XFI XFI XFI)
505 1988 1 |   X;
506 1989 1 |
507 1990 1 | BIND
508 1991 1 |   ERR_SEVERITY = UPLIT BYTE(ERROR_LIST) : VECTOR [256, BYTE];
509 1992 1 |
510 1993 1 | UNDECLARE XQUOTE
511 1994 1 |   ERR;
512 1995 1 |
513 1996 1 | |
514 1997 1 | | Define the system handling option for each error.
515 1998 1 | |
516 1999 1 | |
517 2000 1 | MACRO
518 M 2001 1 |   ERR (NUMBER, CODE, TEXT, SEVERITY, SYSTEM_HANDLING) =
519 M 2002 1 |     (XIF (XIDENTICAL (SYSTEM_HANDLING, CONT)) XTHEN K SYS CONT XELSE
520 M 2003 1 |       XIF (XIDENTICAL (SYSTEM_HANDLING, EXIT)) XTHEN K SYS EXIT XELSE
521 M 2004 1 |       XIF (XIDENTICAL (SYSTEM_HANDLING, RESTART)) XTHEN K_SYS_RESTART XELSE
522 M 2005 1 |       257 XFI XFI XFI)
523 2006 1 |   X;
524 2007 1 |
525 2008 1 | BIND
526 2009 1 |   ERR_SYSTEM = UPLIT BYTE(ERROR_LIST) : VECTOR [256, BYTE];
527 2010 1 |
528 2011 1 | UNDECLARE XQUOTE
529 2012 1 |   ERR;
530 2013 1 |
531 2014 1 | !<BLF/PAGE>
```

```
533 2015 1 1+ The following field set represents an item pushed onto the
534 2016 1 error stack. It contains the entire state of the error system.
535 2017 1 It is used when it is necessary to save the error state to run
536 2018 1 the compiler to compile an immediate mode statement.
537 2019 1
538 2020 1
539 2021 1
540 2022 1 FIELD
541 2023 1   PUSH_ITEM =
542 2024 1   SET
543 2025 1     PUSHSA_NEXT = [0, 0, %BPVAL, 0],      ! Next item
544 2026 1     PUSHSA_PREV = [4, 0, %BPVAL, 0],    ! Previous item
545 2027 1     PUSHSL_ERRFLG = [8, 0, %BPVAL, 0],  ! 1 = error in progress
546 2028 1     PUSHST_ERN = [12, 0, 0, 0],         ! Module name of error
547 2029 1     PUSHSL_ERR = [20, 0, %BPVAL, 0],    ! Error number
548 2030 1     PUSHSL_ERL = [24, 0, %BPVAL, 0],    ! Line number of error
549 2031 1     PUSHSL_HGH_LVL = [28, 0, %BPVAL, 0], ! Level for RESUME
550 2032 1     PUSHSA_HGH_FMP = [32, 0, %BPADDR, 0], ! Frame for RESUME
551 2033 1     PUSHSL_ACC_LVL = [36, 0, %BPVAL, 0], ! Level for RESUME-line
552 2034 1     PUSHSL_UNW_CNT = [40, 0, %BPVAL, 0], ! Amount to unwind at top
553 2035 1     PUSHSL_SYS_ERR = [44, 0, %BPVAL, 0], ! 1 = "fatal fatal" error
554 2036 1     PUSHSL_GONE_BAK = [48, 0, %BPVAL, 0], ! 1 = ON ERROR GO BACK
555 2037 1     PUSHSA_CUR_IN = [52, 0, %BPADDR, 0], ! Restart PC
556 2038 1     PUSHSL_GOING_BACK = [56, 0, %BPVAL, 0], ! Restart flag
557 2039 1     PUSHSA_RESTART = [60, 0, %BPADDR, 0], ! Real restart PC
558 2040 1     TES;
559 2041 1
560 2042 1 LITERAL
561 2043 1   PUSH$K_LENGTH = 64;      ! Number of bytes to allocate
562 2044 1
```



```
564 2045 1 GLOBAL ROUTINE BAS$$SIGNAL (
565 2046 1     ERR_CODE
566 2047 1     ) : NOVALUE =
567 2048 1
568 2049 1
569 2050 1 **
570 2051 1 FUNCTIONAL DESCRIPTION:
571 2052 1     Signal an error for BASIC-PLUS-2/VAX. The argument is the
572 2053 1     BASIC-PLUS-2 error code.
573 2054 1
574 2055 1 FORMAL PARAMETERS:
575 2056 1
576 2057 1     ERR_CODE.r.l.v The BASIC-PLUS-2 error code. The codes and
577 2058 1     their meanings are listed in file BASERRMSG.REQ.
578 2059 1
579 2060 1 IMPLICIT INPUTS:
580 2061 1
581 2062 1     NONE
582 2063 1
583 2064 1 IMPLICIT OUTPUTS:
584 2065 1
585 2066 1     NONE
586 2067 1
587 2068 1 ROUTINE VALUE:
588 2069 1
589 2070 1     NONE
590 2071 1
591 2072 1 COMPLETION CODES:
592 2073 1
593 2074 1     NONE
594 2075 1
595 2076 1 SIDE EFFECTS:
596 2077 1
597 2078 1     May never return to the caller.
598 2079 1
599 2080 1 --
600 2081 1
601 2082 1 BEGIN
602 2083 1
603 2084 1 LOCAL
604 2085 1     VAX_11_COND_VAL : BLOCK [4, BYTE]; ! 32-bit VAX/VMS condition value
605 2086 1
606 2087 1     VAX_11_COND_VAL = BAS$$COND_VAL (.ERR_CODE);
607 2088 1
608 2089 1     The line number, module name and function name are added in
609 2090 1     BAS$HANDLER for each level that this signal goes through.
610 2091 1
611 2092 1     LIB$SIGNAL (.VAX_11_COND_VAL);
612 2093 1     END;
! of BAS$$SIGNAL
```

```
.TITLE BASERROR
.IDENT \1-074\
```

```
.PSECT _BAS$DATA,NOEXE, PIC.2
```

```
00# 00000 BAS$T_ERN::
```


P. AAA
P. AAB

```

.EXTRN LIBSMATCH_COND, LIBSSIGNAL
.EXTRN LIBSSTOP, SYSSUNWIND
.EXTRN LIBSFIXUP_FLT, LIBSGET_VM
.EXTRN LIBSFREE_VM, STR$CONCAT
.EXTRN STR$COPY_R, STR$COPY_DX
.EXTRN SYSSGETMSG, BASSINIT_ONERR
.EXTRN BASS$RESTART_IO
.EXTRN BASS$PUR_IO_ERR
.EXTRN OTSS$TERM_IO, BASS$SUNWIND
.EXTRN BASS$SUNWIND_IO, BASS$HANDLER
.EXTRN BASSK_FAC_NO, BASSK_RESNO_ERR
.EXTRN BASSK_ILLRESSUB
.EXTRN BASSK_DIVBY_ZER
.EXTRN BASSK_IMASQOROO
.EXTRN BASSK_ILLARGLOG
.EXTRN BASSK_INTERR, BASSK_MEMMANVIO
.EXTRN BASSK_FLOPOIERR
.EXTRN BASSK_SUBOUTRAN
.EXTRN BASSK_MAXMEMEXC
.EXTRN BASSK_ILLBYTCOU
.EXTRN BASS_ON_CHAFIL, BASSK_PROLOSSOR
.EXTRN BASSK_STRTOOLON
.EXTRN BASSK_DECERR, BASSK_IMPERRHAN
.EXTRN MTHS_SQROONEG, MTHS_LOGZERNEG
.EXTRN MTHS_FLOOVEMAT, MTHS_UNDEXP
.EXTRN MTHS_SIGLOSMAT, STR$INSVIRMEM
.EXTRN STR$DIVBY_ZER, STR$STRTOOLON
.EXTRN OTSS_IO_CONCLO, OTSS$A_CUR_LUB

```

```

.ENTRY  BAS$$SIGNAL, Save nothing          : 2045
PUSHL   ERR_CODE                          : 2087
CALLS   #1, -BAS$$COND VAL
PUSHL   VAX_11 COND VAL
CALLS   #1, -LIB$$SIGNAL
RET

```

613 2094 1

```

615 2095 1 GLOBAL ROUTINE BAS$$STOP (
616 2096 1     ERR_CODE
617 2097 1     ) : NOVALUE =
618 2098 1
619 2099 1
620 2100 1 **
621 2101 1 FUNCTIONAL DESCRIPTION:
622 2102 1     Signal an error for BASIC-PLUS-2/VAX. The argument is the
623 2103 1     BASIC-PLUS-2 error code.
624 2104 1
625 2105 1 FORMAL PARAMETERS:
626 2106 1
627 2107 1     ERR_CODE.rl.v The BASIC-PLUS-2 error code. The codes and
628 2108 1     their meanings are listed in file BASERRMSG.REQ.
629 2109 1     The severity must be ERROR or SEVERE ERROR.
630 2110 1
631 2111 1 IMPLICIT INPUTS:
632 2112 1
633 2113 1     NONE
634 2114 1
635 2115 1 IMPLICIT OUTPUTS:
636 2116 1
637 2117 1     NONE
638 2118 1
639 2119 1 ROUTINE VALUE:
640 2120 1
641 2121 1     NONE
642 2122 1
643 2123 1 COMPLETION CODES:
644 2124 1
645 2125 1     NONE
646 2126 1
647 2127 1 SIDE EFFECTS:
648 2128 1
649 2129 1     Never returns to the caller.
650 2130 1
651 2131 1 --
652 2132 1
653 2133 2 BEGIN
654 2134 2
655 2135 2 LOCAL
656 2136 2     VAX_11_COND_VAL : BLOCK [4, BYTE]; ! 32-bit VAX/VMS condition value
657 2137 2
658 2138 2     VAX_11_COND_VAL = BAS$$COND_VAL (.ERR_CODE);
659 2139 2
660 2140 2 The line number, module name and function name are added in
661 2141 2 BAS$HANDLER for each level that this signal goes through.
662 2142 2
663 2143 2 LIB$STOP (.VAX_11_COND_VAL);
664 2144 2 END; ! of BAS$$STOP
```

```
04 0000 00000
AC DD 00002
```

```
ENTRY BAS$$STOP, Save nothing
PUSHL ERR_CODE
```

```
: 2095
: 2138
```

BASERROR
1-074

M 8
16-Sep-1984 00:23:13 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 11:54:56 [BASRTL.SRC]BASERROR.B32;1

Page 17
(6)

0000V CF
00000000G 00

01 FB 00005
50 DD 0000A
01 FB 0000C
04 00013

CALLS #1, BAS\$COND VAL
PUSHL VAX_11 COND VAL
CALLS #1, LIB\$STOP
RET

: 2143
: 2144

; Routine Size: 20 bytes. Routine Base: _BAS\$CODE + 0214

; 665 2145 1


```
667 2146 1 GLOBAL ROUTINE BAS$COND_VAL (      ! Compute condition value
668 2147 1   ERR_CODE                        ! BASIC error code
669 2148 1   ) =
670 2149 1
671 2150 1  --
672 2151 1  FUNCTIONAL DESCRIPTION:
673 2152 1
674 2153 1      Convert a BASIC error code to its 32-bit VAX/VMS error code.
675 2154 1      Conversion is done by copying the BASIC error number to the
676 2155 1      code field, setting the severity field based on the entry in
677 2156 1      the severity table for the code, and setting the facility
678 2157 1      to BAS$K_FAC_NO. The facility specific bit is also set.
679 2158 1
680 2159 1  FORMAL PARAMETERS:
681 2160 1
682 2161 1      ERR_CODE.r1.v  The BASIC-PLUS-2 error code. The codes and
683 2162 1                      their meanings are listed in file BASERRMSG.REQ.
684 2163 1
685 2164 1  IMPLICIT INPUTS:
686 2165 1
687 2166 1      NONE
688 2167 1
689 2168 1  IMPLICIT OUTPUTS:
690 2169 1
691 2170 1      NONE
692 2171 1
693 2172 1  ROUTINE VALUE:
694 2173 1
695 2174 1      The 32-bit VAX/VMS error code.
696 2175 1
697 2176 1  COMPLETION CODES:
698 2177 1
699 2178 1      NONE
700 2179 1
701 2180 1  SIDE EFFECTS:
702 2181 1
703 2182 1      NONE
704 2183 1
705 2184 1  --
706 2185 1
707 2186 2  BEGIN
708 2187 2
709 2188 2  LOCAL
710 2189 2      RESULT : BLOCK [4, BYTE];      ! 32-bit VAX/VMS condition value
711 2190 2
712 2191 2  RESULT = 0;
713 2192 2  RESULT [ST$BV_SEVERITY] = (IF (.ERR_CODE GTRU 255) THEN ST$BK_INFO ELSE .ERR_SEVERITY [.ERR_CODE]);
714 2193 2  RESULT [ST$BV_CODE] = .ERR_CODE;
715 2194 2  RESULT [ST$BV_FAC_SP] = 1;
716 2195 2  RESULT [ST$BV_FAC_NO] = BAS$K_FAC_NO;
717 2196 2  RETURN (.RESULT);
718 2197 1  END;                                ! of BAS$COND_VAL
```

16-Sep-1984 00:23:13 VAX-11 B11sg-32 V4.0-742
14-Sep-1984 11:54:56 [BASRTL.SRC]BASERROR.B32:1

```

.ENTRY      BAS$COND_VAL, Save nothing
CLRL       RESULT
CML        ERR_CODE, #255
BLEQU     1$
MOVL      #3, R0
BRB       2$
MOVAB     ERR_SEVERITY, R0
MOVZBL    @ERR_CODE[R0], R0
INSV      R0, R0, #3, RESULT
INSV      ERR_CODE, #3, #12, RESULT
DISW2     #32768, RESULT
INSV      #BAS$K_FAC_NO, #16, #12, RESULT
MOVL      RESULT, R0
RET

```

2146
2191
2192

2193
2194
2195
2196
2197

: 719 2198 1

```

721 2199 1 ROUTINE PC_TO_LINE_NO (      ! Convert PC to Line number
722 2200 1     FMP,                      ! Frame containing line
723 2201 1     PC                      ! PC to be converted
724 2202 1     ) =
725 2203 1
726 2204 1
727 2205 1 **
728 2206 1 FUNCTIONAL DESCRIPTION:
729 2207 1     Compute the BASIC line number corresponding to a PC value.
730 2208 1
731 2209 1 FORMAL PARAMETERS:
732 2210 1
733 2211 1     FMP.ra.v      Address of the frame from which we want the
734 2212 1                     line number.
735 2213 1     PC.rlu.v    The program counter corresponding to the
736 2214 1                     line number. If no exact match is found, use
737 2215 1                     the next lower PC value.
738 2216 1
739 2217 1 IMPLICIT INPUTS:
740 2218 1
741 2219 1     The (delta PC, line number) table, pointed to by the FCD
742 2220 1     for the main procedure.
743 2221 1
744 2222 1 IMPLICIT OUTPUTS:
745 2223 1
746 2224 1     NONE
747 2225 1
748 2226 1 ROUTINE VALUE:
749 2227 1
750 2228 1     The line number, as a 32-bit binary value.
751 2229 1
752 2230 1 COMPLETION CODES:
753 2231 1
754 2232 1     NONE
755 2233 1
756 2234 1 SIDE EFFECTS:
757 2235 1
758 2236 1     NONE
759 2237 1
760 2238 1 --
761 2239 1
762 2240 1 BEGIN
763 2241 1
764 2242 1 MAP
765 2243 1     FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);
766 2244 1
767 2245 1 LOCAL
768 2246 1     BSF$A MAJOR STG : REF BLOCK [0, BYTE] FIELD (BSF$MAJOR_FRAME),
769 2247 1     PC_DELTA_TABLE : REF VECTOR,
770 2248 1     SEARCH_ARG;
771 2249 1
772 2250 1
773 2251 1 **
774 2252 1 If the PC is zero, the cell that held it must not have been
775 2253 1 set up. This means that we are trying to find the PC for
776 2254 1 a routine in which the first statement has not yet started
777 2255 1 execution. Return a zero to indicate this.

```



```
778 2256 IF (.PC EQLA 0) THEN RETURN (0);
779 2257
780 2258
781 2259
782 2260 Load the pointer to the major procedure's frame.
783 2261
784 2262 BSFSA_MAJOR_STG = .FMP [BSFSA_BASE_R11];
785 2263
786 2264 Get a pointer to the (PC delta, line number) table.
787 2265 This requires skipping over the text of the module name.
788 2266
789 2267 PC_DELTA_TABLE = .BSFSA_MAJOR_STG [BSFSA_PROC_INFO] +
790 2268 ((.BSFSA_MAJOR_STG [BSFSA_PROC_INFO]) AND 255) + 1;
791 2269
792 2270 Compute the PC relative to the beginning of the code.
793 2271
794 2272 SEARCH_ARG = .PC - .BSFSA_MAJOR_STG [BSFSA_CODE_BEG];
795 2273
796 2274 Search the table. If an exact match cannot be found, use the
797 2275 line number just before the PC.
798 2276
799 2277
800 2278 DECR TABLE_INDEX FROM .PC_DELTA_TABLE [0] TO 1 DO
801 2279 BEGIN
802 2280
803 2281 IF (((.PC_DELTA_TABLE [.TABLE_INDEX]) AND 65535) LEQU .SEARCH_ARG)
804 2282 THEN
805 2283 RETURN ((.PC_DELTA_TABLE [.TABLE_INDEX]) AND 65535);
806 2284
807 2285 END;
808 2286
809 2287
810 2288 We get here only if the number cannot be found in the table. This
811 2289 means that the PC was stored by a fake line number before the first
812 2290 real line number. This is so unreasonable that it is more likely
813 2291 due to a bug in either the compiler or the RTL. To make the problem
814 2292 more visible, return a -1.
815 2293
816 2294 RETURN (-1);
817 2295 END;
```

! of PC_TO_LINE_NO

```
000C 00000 PC_TO_LINE_NO:
08 AC D5 00002 TSTL PC
3C 13 00005 BEQL 3$
50 04 AC D0 00007 MOVL FMP, R0
50 F4 A0 D0 0000B MOVL -12(R0), BSFSA_MAJOR_STG
51 00AB D0 9A 0000F MOVZBL @171(BSFSA_MAJOR_STG), R1
51 00AB C0 C0 00014 ADDL2 171(BSFSA_MAJOR_STG), R1
53 08 AC 0083 51 D6 00019 INCL PC_DELTA_TABLE
50 01 C1 00022 SUBL3 13T(BSFSA_MAJOR_STG), PC, SEARCH_ARG
14 11 00026 ADDL3 #1, (PC_DELTA_TABLE), TABLE_INDEX
BRB 2$
```

```
2199
2257
2262
2268
2272
2281
```

BASERROR
1-074

M 8
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

VAX-11 Bliss-32 V4.0-742
[BASRTL.SRC]BASERROR.B32;1

Page 22
(8)

53	52	6140	F0	8F	78	00028	1\$:	ASHL	#-16, (PC DELTA TABLE)[TABLE_INDEX], R2	
	52	10		00	ED	0002E		CMPZV	#0, #16, R2, SEARCH_ARG	
				07	1A	00033		BGTRU	2\$	
		50	6140	DF	00035			PUSHAL	(PC DELTA TABLE)[TABLE_INDEX]	2283
				9E	3C	00038		MOVZWL	@(SP)+, R0	
		E9		04	0003B			RET		
		50	50	F5	0003C	2\$:		SOBGTR	TABLE_INDEX, 1\$	2278
				01	CE	0003F		MNEGL	#1, R0	2294
				04	00042			RET		
			50	D4	00043	3\$:		CLRL	R0	2295
				04	00045			RET		

; Routine Size: 70 bytes, Routine Base: _BAS\$CODE + 0262

```
819 2296 1 GLOBAL ROUTINE BAS$$LINE (      ! Get current line number
820 2297 1     FMP                          ! Current frame
821 2298 1     ) =
822 2299 1
823 2300 1 ++
824 2301 1 FUNCTIONAL DESCRIPTION:
825 2302 1
826 2303 1     Compute the number of the line now in execution. At the
827 2304 1     beginning of each line, the BASIC compiler issues
828 2305 1
829 2306 1     MOVAB    -3(PC),BSF$A_MARK(FP)
830 2307 1
831 2308 1     which stores the PC of the MOVAB. This routine then
832 2309 1     scans the PC delta table (also produced by the compiler)
833 2310 1     to find the correct line number.
834 2311 1
835 2312 1 FORMAL PARAMETERS:
836 2313 1
837 2314 1     FMP.ra.v      Address of the frame from which we want the
838 2315 1                  line number.
839 2316 1
840 2317 1 IMPLICIT INPUTS:
841 2318 1
842 2319 1     NONE
843 2320 1
844 2321 1 IMPLICIT OUTPUTS:
845 2322 1
846 2323 1     NONE
847 2324 1
848 2325 1 ROUTINE VALUE:
849 2326 1
850 2327 1     The line number, as a 32-bit binary value.
851 2328 1
852 2329 1 COMPLETION CODES:
853 2330 1
854 2331 1     NONE
855 2332 1
856 2333 1 SIDE EFFECTS:
857 2334 1
858 2335 1     NONE
859 2336 1
860 2337 1 --
861 2338 1
862 2339 1 BEGIN
863 2340 1
864 2341 1 MAP
865 2342 1     FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);
866 2343 1
867 2344 1 RETURN (PC_TO_LINE_NO (.FMP, .FMP [BSF$A_MARK]));
868 2345 1 END;                                ! of BAS$$LINE
```

```
50      04      0000 00000      .ENTRY BAS$$LINE, Save nothing
          AC      DO 00002      MOVL    FMP, R0
```

```
: 2296
: 2344
```



```

8 9
16-Sep-1984 00:23:13 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 11:54:56 [BASRTL.SRC]BASERROR.B32;1

```

Page 24
(9)

		FC	A0	DD	00006	PUSHL	-4(R0)
			50	DD	00009	PUSHL	R0
AB	AF		02	FB	0000B	CALLS	#2, PC_TO_LINE_NO
				04	0000F	RET	

2345

; Routine Size: 16 bytes, Routine Base: _BAS\$CODE + 02AB

```
870 2346 1 ROUTINE BAS$FUNCTION (
871 2347 1 FMP
872 2348 1 ) =
873 2349 1
874 2350 1 **
875 2351 1 FUNCTIONAL DESCRIPTION:
876 2352 1
877 2353 1 Get the name of the function now in execution. It is
878 2354 1 obtained from the BSF$A_PROC_ID field of the frame.
879 2355 1 The format depends on the frame type. For the types
880 2356 1 which have no name, the "name" is returned as a line
881 2357 1 number, and the caller expects this. In other cases,
882 2358 1 the name is returned as a pointer to a counted string.
883 2359 1
884 2360 1 FORMAL PARAMETERS:
885 2361 1
886 2362 1 FMP.ra.v Address of the frame from which we want the
887 2363 1 function name.
888 2364 1
889 2365 1 IMPLICIT INPUTS:
890 2366 1
891 2367 1 NONE
892 2368 1
893 2369 1 IMPLICIT OUTPUTS:
894 2370 1
895 2371 1 NONE
896 2372 1
897 2373 1 ROUTINE VALUE:
898 2374 1
899 2375 1 The function name, as a pointer to a counted string for
900 2376 1 main procedures, subprograms, external functions, DEFs
901 2377 1 and DEFs. The name is returned as a 32-bit line number
902 2378 1 for GOSUBs and condition handlers.
903 2379 1
904 2380 1 COMPLETION CODES:
905 2381 1
906 2382 1 NONE
907 2383 1
908 2384 1 SIDE EFFECTS:
909 2385 1
910 2386 1 NONE
911 2387 1
912 2388 1 --
913 2389 1
914 2390 2 BEGIN
915 2391 2
916 2392 2 MAP
917 2393 2 FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);
918 2394 2
919 2395 2 RETURN (CASE .FMP [BSF$B_PROC_CODE] FROM BSF$K_PROC_MAIN TO BSF$K_PROC_IOL OF
920 2396 2 SET
921 2397 2 [BSF$K_PROC_MAIN, BSF$K_PROC_SUB, BSF$K_PROC_EXTF, BSF$K_PROC_DEF, BSF$K_PROC_DEFS] :
922 2398 2 .FMP [BSF$A_PROC_ID];
923 2399 2 [BSF$K_PROC_GOSB, BSF$K_PROC_ONER, BSF$K_PROC_IOL] :
924 2400 2 PC-TO LINE NO (.FMP, .FMP [BSF$A_PROC_ID]);
925 2401 2 [OUTRANGE] : 0;
926 2402 2 TES);
```

BASSERRON
1-074

16-Sep-1984 00:23:13 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 11:54:56 [BASRTL.SRC]BASERROR.B32:1

Page 26
(10)

: 927 2403 1 END;

! of BASSFUNCTION

0000 00000 BASS\$FUNCTION:										
0013	07	50	04	AC	D0	00002		.WORD	Save nothing	2346
0018	01	01	E5	A0	8F	00006		MOVL	FMP, R0	2395
	0013	0013		0013		00008	1\$:	CASEB	-27(R0), #1, #7	
	0018	0018		0013		00013		.WORD	2\$-1\$,-	
									2\$-1\$,-	
									2\$-1\$,-	
									2\$-1\$,-	
									2\$-1\$,-	
									3\$-1\$,-	
									3\$-1\$,-	
									3\$-1\$,-	
									3\$-1\$	
				50	D4	0001B		CLRL	R0	
				04	0001D			RET		
		50	E8	A0	D0	0001E	2\$:	MOVL	-24(R0), R0	2398
				04	00022			RET		
			E8	A0	DD	00023	3\$:	PUSHL	-24(R0)	2400
				50	DD	00026		PUSHL	R0	
	FF7D	CF		02	FB	00028		CALLS	#2, PC_TO_LINE_NO	2403
				04	0002D			RET		

: Routine Size: 46 bytes, Routine Base: _BASSCODE + 02B8


```
929 2404 1 GLOBAL ROUTINE BAS$$MODULE (          ! Get current module name
930 2405 1     FMP                                ! Current frame
931 2406 1     ) =
932 2407 1
933 2408 1
934 2409 1 **
935 2410 1 FUNCTIONAL DESCRIPTION:
936 2411 1     Get the name of the module now in execution. It is
937 2412 1     obtained from the BSF$A_PROC_ID field of the frame.
938 2413 1     It is returned as a pointer to a counted string.
939 2414 1
940 2415 1 FORMAL PARAMETERS:
941 2416 1
942 2417 1     FMP.ra.v      Address of the frame from which we want the
943 2418 1                  module name.
944 2419 1
945 2420 1 IMPLICIT INPUTS:
946 2421 1
947 2422 1     NONE
948 2423 1
949 2424 1 IMPLICIT OUTPUTS:
950 2425 1
951 2426 1     NONE
952 2427 1
953 2428 1 ROUTINE VALUE:
954 2429 1
955 2430 1     The module name, as a pointer to a counted string.
956 2431 1
957 2432 1 COMPLETION CODES:
958 2433 1
959 2434 1     NONE
960 2435 1
961 2436 1 SIDE EFFECTS:
962 2437 1
963 2438 1     NONE
964 2439 1
965 2440 1 --
966 2441 1
967 2442 1 BEGIN
968 2443 1
969 2444 1 MAP
970 2445 1     FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);
971 2446 1
972 2447 1 LOCAL
973 2448 1     BSF$A_MAJOR_STG : REF BLOCK [0, BYTE] FIELD (BSF$MAJOR_FRAME);
974 2449 1
975 2450 1
976 2451 1 * Load the pointer to the major procedure's frame.
977 2452 1
978 2453 1     BSF$A_MAJOR_STG = .FMP [BSF$A_BASE_R11];
979 2454 1
980 2455 1 * Its procedure information starts with the name of the module.
981 2456 1
982 2457 1 RETURN (.BSF$A_MAJOR_STG [BSF$A_PROC_INFO]);
983 2458 1 END;                                ! of BAS$$MODULE
```

BASERROR
1-074

F 9
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

YAX-11 Bliss-32 V4.0-742
[BASRTL.SRC]BASERROR.B32:1

Page 28
(11)

			0000	00000
50	04	AC	DO	00002
50	F4	AO	DO	00006
50	00AB	CO	DO	0000A
			04	0000F

.ENTRY BAS\$MODULE, Save nothing
MOVL FMP, RO
MOVL -12(RO), BSF\$A_MAJOR_STG
MOVL 171(BSF\$A_MAJOR_STG), RO
RET

: 2404
: 2453
: 2457
: 2458

; Routine Size: 16 bytes. Routine Base: _BAS\$CODE + 02E6

```

985 2459 1 ROUTINE HANDLER_HANDLER (
986 2460 1     SIG,
987 2461 1     MECH,
988 2462 1     ENBL
989 2463 1 ) =
990 2464 1
991 2465 1
992 2466 1
993 2467 1
994 2468 1
995 2469 1
996 2470 1
997 2471 1
998 2472 1
999 2473 1
1000 2474 1
1001 2475 1
1002 2476 1
1003 2477 1
1004 2478 1
1005 2479 1
1006 2480 1
1007 2481 1
1008 2482 1
1009 2483 1
1010 2484 1
1011 2485 1
1012 2486 1
1013 2487 1
1014 2488 1
1015 2489 1
1016 2490 1
1017 2491 1
1018 2492 1
1019 2493 1
1020 2494 1
1021 2495 1
1022 2496 1
1023 2497 1
1024 2498 1
1025 2499 1
1026 2500 1
1027 2501 1
1028 2502 1
1029 2503 1
1030 2504 1
1031 2505 1
1032 2506 1
1033 2507 1
1034 2508 1
1035 2509 1
1036 2510 1
1037 2511 1
1038 2512 1
1039 2513 1
1040 2514 1
1041 2515 1

ROUTINE HANDLER_HANDLER (
    SIG,
    MECH,
    ENBL
) =

**
FUNCTIONAL DESCRIPTION:
    Handle the unwind which is likely to be done to BAS$HANDLER
    by freeing its heap storage. The length and address of the
    heap storage are passed in the third argument to this routine.

FORMAL PARAMETERS:
    SIG.rl.a      Address of the signal vector. This contains
                  the condition.
    MECH.rl.a     Address of the mechanism vector. This contains
                  the status of the frame that signalled.
    ENBL.rl.a     Address of the enable vector. This contains
                  pointers to the variables used to free the
                  heap storage for BAS$HANDLER.

IMPLICIT INPUTS:
    NONE

IMPLICIT OUTPUTS:
    NONE

ROUTINE VALUE:
    NONE

COMPLETION CODES:
    Always SSS_RESIGNAL, but this is ignored when we are
    unwinding.

SIDE EFFECTS:
    May call LIB$FREE_VM to return storage to the free pool.

--
BEGIN
MAP
    SIG : REF VECTOR,
    MECH : REF VECTOR,
    ENBL : REF VECTOR;
    | signal vector
    | mechanism vector
    | enable vector

**
First check for the unwinding condition. If it is not, resignal.

```



```
1042 2516 2 IF ( NOT (LIBSMATCH_COND (SIG [1], %REF (SS%_UNWIND))) ) THEN RETURN (SS%_RESIGNAL);
1043 2517
1044 2518
1045 2519 + We are unwinding. If any heap storage has been allocated, free it.
1046 2520
1047 2521
1048 2522 IF ( ( (ENBL [2]) NEQA 0 ) THEN LIB$FREE_VM (%REF ( (ENBL [1]) * %UPVAL ), ENBL [2]);
1049 2523
1050 2524
1051 2525 + All done.
1052 2526
1053 2527 RETURN (SS%_RESIGNAL);
1054 2528 END;
```

! of HANDLER_HANDLER

```
0000 00000 HANDLER_HANDLER:
      7E      0920 8F 3C 00002 .WORD Save nothing
      5E DD 00007 MOVZWL #2336, -(SP)
      04 C1 00009 PUSHL SP
      02 FB 0000E ADDL3 #4, SIG, -(SP)
      50 E9 00015 CALLS #2, LIB$MATCH_COND
      0C AC D0 00018 BLBC R0, 1$
      08 B0 D5 0001C MOVL ENBL, R0
      13 13 0001F TSTL @8(R0)
      08 A0 D0 00021 BEQL 1$
      02 78 00024 PUSHL 8(R0)
      04 AE 9F 0002A ASHL #2, @4(R0), 4(SP)
      02 FB 0002D PUSHAB 4(SP)
      50      0918 8F 3C 00034 1$: CALLS #2, LIB$FREE_VM
      04 00039 MOVZWL #2528, R0
      RET
```

; Routine Size: 58 bytes. Routine Base: _BAS\$CODE + 02F6

```
2459
2516
2522
2527
2528
```

```
1056 2529 1 ROUTINE RESTART : RESTART_LINK NOVALUE =
1057 2530 1
1058 2531 1
1059 2532 1 **
1060 2533 1 FUNCTIONAL DESCRIPTION:
1061 2534 1 This is a short routine which restores SP and branches to the
1062 2535 1 user's code. It has to be a routine so its address can be
1063 2536 1 passed to SYS$UNWIND.
1064 2537 1 Before branching to the user's code it POPs any I/O that may
1065 2538 1 be in progress.
1066 2539 1
1067 2540 1 FORMAL PARAMETERS:
1068 2541 1
1069 2542 1 NONE
1070 2543 1
1071 2544 1 IMPLICIT INPUTS:
1072 2545 1
1073 2546 1 BASSA_RESTART.ra The PC to branch to, which will be the
1074 2547 1 the first instruction of a line.
1075 2548 1
1076 2549 1 IMPLICIT OUTPUTS:
1077 2550 1
1078 2551 1 NONE
1079 2552 1
1080 2553 1 ROUTINE VALUE:
1081 2554 1
1082 2555 1 NONE
1083 2556 1
1084 2557 1 COMPLETION CODES:
1085 2558 1
1086 2559 1 NONE
1087 2560 1
1088 2561 1 SIDE EFFECTS:
1089 2562 1
1090 2563 1 Never returns to its 'caller'
1091 2564 1
1092 2565 1 --
1093 2566 1
1094 2567 1 BEGIN
1095 2568 1
1096 2569 1 REGISTER
1097 2570 1 FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);
1098 2571 1
1099 2572 1 BUILTIN
1100 2573 1 FP;
1101 2574 1 SP;
1102 2575 1
1103 2576 1 FMP = .FP;
1104 2577 1 BASS$UNWIND IO (.FMP); ! POP this frame's I/O
1105 2578 1 SP = .FMP [BSF$A BASE_SP]; ! Restore SP
1106 2579 1 SP = .SP - %UPVAL;
1107 2580 1 .SP = .BASSA_RESTART; ! Specify place to go
1108 2581 1 RETURN; ! Go there, in effect.
1109 2582 1 END; ! end of RESTART
```

BASERROR
1-074

J 9
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

VAX-11 B11sg-32 V4.0-742
[BASRTL.SRC]BASERROR.B32;1

Page 32
(13)

52	5D	DD	00000	RESTART:MOVL	FP, FMP	: 2576
00000000G	52	DD	00003	PUSHL	FMP	: 2577
00	01	FB	00005	CALLS	#1, BAS\$UNWIND_ID	: 2578
5E	A2	DD	0000C	MOVL	-8(FMP), SP	: 2579
5E	04	C2	00010	SUBL2	#4, SP	: 2580
6E 00000000'	EF	DD	00013	MOVL	BAS\$A_RESTART, (SP)	: 2582
	05	DD	0001A	RSB		

; Routine Size: 27 bytes, Routine Base: _BAS\$CODE + 0330


```
1111 2583 1 ROUTINE RESTART_IO : RESTART_LINK NOVALUE =
1112 2584 1
1113 2585 1
1114 2586 1 ++
1115 2587 1 FUNCTIONAL DESCRIPTION:
1116 2588 1 This is a short routine to call BAS$$RESTART_IO when unwinding
1117 2589 1 to the beginning of an I/O list. No I/O popping is done.
1118 2590 1
1119 2591 1 FORMAL PARAMETERS:
1120 2592 1
1121 2593 1 NONE
1122 2594 1
1123 2595 1 IMPLICIT INPUTS:
1124 2596 1
1125 2597 1 Gets the PC to branch to from BAS$$RESTART_IO.
1126 2598 1
1127 2599 1 IMPLICIT OUTPUTS:
1128 2600 1
1129 2601 1 NONE
1130 2602 1
1131 2603 1 ROUTINE VALUE:
1132 2604 1
1133 2605 1 NONE
1134 2606 1
1135 2607 1 COMPLETION CODES:
1136 2608 1
1137 2609 1 NONE
1138 2610 1
1139 2611 1 SIDE EFFECTS:
1140 2612 1
1141 2613 1 Never returns to its 'caller'
1142 2614 1
1143 2615 1 --
1144 2616 1
1145 2617 2 BEGIN
1146 2618 2
1147 2619 2 REGISTER
1148 2620 2 FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);
1149 2621 2
1150 2622 2 BUILTIN
1151 2623 2 FP;
1152 2624 2 SP;
1153 2625 2
1154 2626 2 FMP = .FP;
1155 2627 2 SP = .FMP [BSF$A BASE_SP]; ! Restore SP
1156 2628 2 SP = .SP - %UPVAL;
1157 2629 2 .SP = BAS$$RESTART_IO (); ! Get place to go
1158 2630 2 RETURN; ! Go there, in effect.
1159 2631 1 END; ! end of RESTART_IO
```

```
50          SD DO 00000 RESTART_IO:
SE          F8 A0 DO 00003          -MOVL FP, FMP
                                         MOVL -8(FMP), SP
```

```
: 2626
: 2627
```

BAS\$ERROR
1-074

L 9
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

VAX-11 Bliss-32 V4.0-742
[BASRTL.SRC]BASERROR.B32;1

Page 34
(14)

00000000G 5E
00
6E

04 C2 00007
00 FB 0000A
50 D0 00011
05 00014

SUBL2 #4, SP
CALLS #0, BAS\$\$RESTART_10
MOVL R0, (SP)
RSB

: 2628
: 2629
: 2631

; Routine Size: 21 bytes. Routine Base: _BAS\$CODE + 0340

```
1161 2632 1 ROUTINE BAS$USER_HAND (      | interface to user's condition handler
1162 2633 1     ERR_CODE,                  | BASIC error code
1163 2634 1     FMP,                      | user's frame
1164 2635 1     LEVEL                    | level to unwind
1165 2636 1     ) =
1166 2637 1
1167 2638 1 **
1168 2639 1 FUNCTIONAL DESCRIPTION:
1169 2640 1
1170 2641 1     Try to pass a SIGNALed condition to the BASIC user's program
1171 2642 1     for processing.
1172 2643 1
1173 2644 1 FORMAL PARAMETERS:
1174 2645 1
1175 2646 1     ERR_CODE.rl.v    The BASIC error code that is being signaled.
1176 2647 1     FMP.rl.v        Pointer to the frame of the BASIC program
1177 2648 1     LEVEL.rl.v      Number of levels to unwind to get to the
1178 2649 1                    current frame of the BASIC program.
1179 2650 1
1180 2651 1 IMPLICIT INPUTS:
1181 2652 1
1182 2653 1     BAS$L_ERRFLG    0 if no error in progress, 1 if an error is
1183 2654 1                    in progress.
1184 2655 1
1185 2656 1 IMPLICIT OUTPUTS:
1186 2657 1
1187 2658 1     BAS$L_ERRFLG    Set to 1 while we are doing error processing.
1188 2659 1     BAS$L_ERL       The line number being executed when the error
1189 2660 1                    occurred.
1190 2661 1     BAS$T_ERN       The name of the module in which the error
1191 2662 1                    occurred.
1192 2663 1     BAS$L_ERR       The BASIC error number now being processed.
1193 2664 1     HIGHEST_LEVEL   If ON ERROR GO BACK, the level to UNWIND to
1194 2665 1                    if a lower level does a RESUME with no line
1195 2666 1                    number.
1196 2667 1     HIGHEST_FMP     If ON ERROR GO BACK, the frame to UNWIND to
1197 2668 1                    if a lower level does a RESUME with no line
1198 2669 1                    number.
1199 2670 1     ACCUM_LEVEL     If ON ERROR GO BACK, the number of levels above
1200 2671 1                    which must be unwound through if a lower
1201 2672 1                    level does a RESUME with a line number.
1202 2673 1     UNWIND_COUNT    If non-zero, the number of levels to UNWIND
1203 2674 1                    when we get back to the top level call of
1204 2675 1                    BAS$HANDLER.
1205 2676 1
1206 2677 1 ROUTINE VALUE:
1207 2678 1
1208 2679 1     USER_HAND_CONT (=0) => The user has processed the error condition,
1209 2680 1     continue from the point of error (or from an unwind). If an
1210 2681 1     unwind is needed, UNWIND_COUNT is set for the highest level
1211 2682 1     handler.
1212 2683 1
1213 2684 1     USER_HAND_BACK (=1) => The user is not prepared to handle
1214 2685 1     the error at this level, but he may be able to handle it at a
1215 2686 1     deeper level. Revert.
1216 2687 1
1217 2688 1     USER_HAND_FAIL (=2) => The user demands system processing of
```

```

1218 2689 1      this error; do not test deeper levels.  Revert and do not call
1219 2690 1      BAS$$USER_HAND again for this error.
1220 2691 1
1221 2692 1      COMPLETION CODES:
1222 2693 1
1223 2694 1      NONE
1224 2695 1
1225 2696 1      SIDE EFFECTS:
1226 2697 1
1227 2698 1      May call user code.
1228 2699 1
1229 2700 1      --
1230 2701 1
1231 2702 2      BEGIN
1232 2703 2
1233 2704 2      LITERAL
1234 2705 2          K_MAJOR = 1,                !A major frame
1235 2706 2          K_MINOR = 0;                !A minor frame
1236 2707 2
1237 2708 2      MAP
1238 2709 2          FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);
1239 2710 2
1240 2711 2      LOCAL
1241 2712 2          MOD_NAME_ADDR,
1242 2713 2          NON_GOSUB_FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD),
1243 2714 2          USER_HAND_VAL,
1244 2715 2          ONER_RESULT,
1245 2716 2          BSF$A_MAJOR_STG : REF BLOCK [0, BYTE] FIELD (BSF$MAJOR_FRAME),
1246 2717 2          BSF$A_MINOR_STG : REF BLOCK [0, BYTE] FIELD (BSF$MINOR_FRAME),
1247 2718 2          NEXT_FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD),
1248 2719 2          THIS_FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD),
1249 2720 2          MAJOR OR MINOR,
1250 2721 2          SEARCH_DONE;
1251 2722 2
1252 2723 2      BUILTIN
1253 2724 2          FP;
1254 2725 2
1255 2726 2      +
1256 2727 2      If the severity is "error" or "warning", let the compiled code
1257 2728 2      intercept the error, if it has requested to do so.  Note that
1258 2729 2      we must check the severity table rather than the condition value
1259 2730 2      since LIB$STOP forces the severity to "severe error".
1260 2731 2      -
1261 2732 2
1262 2733 2      IF ((.ERR_SEVERITY [.ERR_CODE] NEQ ST$K_ERROR)      !
1263 2734 2          AND (.ERR_SEVERITY [.ERR_CODE] NEQ ST$K_WARNING))
1264 2735 2      THEN
1265 2736 2          RETURN (USER_HAND_FAIL);
1266 2737 2
1267 2738 2      IF (.BAS$L_ERRFLG NEQ 0)
1268 2739 2      THEN
1269 2740 2      +
1270 2741 2      The user has committed an error or said ON ERROR GOTO 0 durring
1271 2742 2      error processing.  Demand system processing.
1272 2743 2      -
1273 2744 2          RETURN (USER_HAND_FAIL);
1274 2745 2

```



```
1275 2746 IF (.BASSL_GOING_BACK NEQ 0)
1276 2747 THEN
1277 2748
1278 2749 * We are one BASIC level deeper in ON ERROR GO BACK processing
1279 2750
1280 2751 BEGIN
1281 2752 BASSL_ERRFLG = 1;
1282 2753 ACCUM_LEVEL = .ACCUM_LEVEL + .LEVEL;
1283 2754 END
1284 2755 ELSE
1285 2756
1286 2757 * This is the first time we have seen this error. Set things up.
1287 2758
1288 2759 BEGIN
1289 2760
1290 2761 *
1291 2762 * only set 'going_back' if this is not a restartable error, i.e.,
1292 2763 * not err=50 (data format error) and not err=52 (illegal number).
1293 2764 * This statement may need further conditionalization if more
1294 2765 * restartable errors are added.
1295 2766
1296 2767 BASSL_GOING_BACK = ( IF (( .ERR_CODE EQL 50 ) OR ( .ERR_CODE EQL 52 ))
1297 2768 THEN 0
1298 2769 ELSE 1 );
1299 2770
1300 2771 BASSL_ERRFLG = 1;
1301 2772 BASSL_CH_CUR_LN = .FMP [BSFSA_MARK];
1302 2773 BASSL_ERR = BASS$LINE (.FMP);
1303 2774 MOD NAME_ADDR = BASS$MODULE (.FMP);
1304 2775 BASSL_ERR [DSC$A_POINTER] = .MOD NAME_ADDR + 1;
1305 2776 BASSL_ERR [DSC$W_LENGTH] = .BLOCK [.MOD NAME_ADDR, 0, 0, 8, 0; 1, BYTE];
1306 2777 BASSL_ERR [DSC$B_CLASS] = DSC$K_CLASS_S;
1307 2778 BASSL_ERR [DSC$B_DTYPE] = DSC$K_DTYPE_T;
1308 2779 BASSL_ERR = .ERR_CODE;
1309 2780 HIGHEST_LEVEL = .LEVEL;
1310 2781 HIGHEST_FMP = .FMP;
1311 2782 ACCUM_LEVEL = .LEVEL;
1312 2783 END;
1313 2784
1314 2785 *
1315 2786 * Fetch the current value of BSFSA_USER_HAND.
1316 2787 * We must first dig back to the first non-GOSUB frame.
1317 2788
1318 2789 NON_GOSUB_FMP = .FMP;
1319 2790 SEARCH_DONE = 0;
1320 2791
1321 2792 DO
1322 2793 BEGIN
1323 2794
1324 2795 IF (.NON_GOSUB_FMP [BSFSA_HANDLER] EQL BASS$HANDLER)
1325 2796 THEN
1326 2797
1327 2798 IF (.NON_GOSUB_FMP [BSF$B_PROC_CODE] NEQ BSF$K_PROC_GOSB) THEN SEARCH_DONE = 1;
1328 2799
1329 2800 IF ( NOT .SEARCH_DONE)
1330 2801 THEN
1331 2802 BEGIN
```

```
1332 2803      NON_GOSUB_FMP = .NON_GOSUB_FMP [BSFSA_SAVED_FP];
1333 2804
1334 2805      IF (.NON_GOSUB_FMP EQLA 0)
1335 2806      THEN
1336 2807          *
1337 2808          We have been unable to find a non-GOSUB frame. This is quite
1338 2809          unreasonable. Force system handling on this error.
1339 2810
1340 2811          RETURN (USER_HAND_FAIL);
1341 2812
1342 2813      END;
1343 2814
1344 2815      END
1345 2816  UNTIL (.SEARCH_DONE);
1346 2817
1347 2818      CASE .NON_GOSUB_FMP [BSF$B_PROC_CODE] FROM BSF$K_PROC_MAIN TO BSF$K_PROC_IOL OF
1348 2819      SET
1349 2820
1350 2821          [BSF$K_PROC_MAIN, BSF$K_PROC_SUB, BSF$K_PROC_EXTF, BSF$K_PROC_DEFS] :
1351 2822          BEGIN
1352 2823              BSF$A_MAJOR_STG = .NON_GOSUB_FMP [BSF$A_BASE_R11];
1353 2824              USER_HAND_VAL = .BSF$A_MAJOR_STG [BSF$A_USER_HAND];
1354 2825              MAJOR_OR_MINOR = K_MAJOR;
1355 2826          END;
1356 2827
1357 2828          [BSF$K_PROC_DEF] :
1358 2829          BEGIN
1359 2830              BSF$A_MINOR_STG = .NON_GOSUB_FMP [BSF$A_BASE_R10];
1360 2831              USER_HAND_VAL = .BSF$A_MINOR_STG [BSF$A_USER_HAND];
1361 2832              MAJOR_OR_MINOR = K_MINOR;
1362 2833          END;
1363 2834
1364 2835          [BSF$K_PROC_IOL] :
1365 2836          BEGIN
1366 2837              USER_HAND_VAL = 0;
1367 2838          END;
1368 2839
1369 2840          [BSF$K_PROC_GOSB, BSF$K_PROC_ONER, OUTRANGE] :
1370 2841              USER_HAND_VAL = 0;
1371 2842          TES;
1372 2843
1373 2844          IF (.USER_HAND_VAL EQL 0)
1374 2845          THEN
1375 2846              *
1376 2847              The user has specified (or defaulted to) system error handling
1377 2848              for any errors. Revert and don't call BAS$$USER_HAND again.
1378 2849
1379 2850              RETURN (USER_HAND_FAIL);
1380 2851
1381 2852          IF (.USER_HAND_VAL EQL 1)
1382 2853          THEN
1383 2854              BEGIN
1384 2855              *
1385 2856              The user has specified ON ERROR GO BACK. Revert but do call
1386 2857              BAS$$USER_HAND again. Note that GOSUBs get unwound one at a
1387 2858              time by this mechanism, but we will make the same decision
1388 2859              each time because the frame is marked for an immediate
```

```
1389 2860 3 ON ERROR GO BACK.
1390 2861
1391 2862     BASSL_ERRFLG = 0;
1392 2863     RETURN (USER_HAND_BACK);
1393 2864     END;
1394 2865
1395 2866
1396 2867     *
1397 2868     The user has specified an entry point in this frame for error
1398 2869     processing. We call his code as a modified GOSUB.
1399 2870     Further system processing depends on how the user's code terminates.
1400 2871     ONER_RESULT = BASSINIT_ONERR (.NON_GOSUB_FMP, .USER_HAND_VAL);
1401 2872
1402 2873     SELECTONEU (.ONER_RESULT) OF
1403 2874     SET
1404 2875
1405 2876     [USER_ERR_RSUMZ] :
1406 2877
1407 2878     *
1408 2879     The condition handler ended with a RESUME with no line number.
1409 2880     Unwind to the frame in which the signal happened and restart
1410 2881     the statement.
1411 2882
1412 2883     BEGIN
1413 2884     BASSL_ERRFLG = 0;
1414 2885     BASSA_RESTART = .BASSA_CH_CUR_LN;
1415 2886     BASSA_CH_CUR_LN = 0;
1416 2887     BASSL_GOING_BACK = 0;
1417 2888     UNWIND_COUNT = .HIGHEST_LEVEL;
1418 2889
1419 2890     *
1420 2891     The compiler should not permit a module to exist with RESUME with no
1421 2892     line number and /NOLINE, but check for that case here and give an
1422 2893     error message.
1423 2894
1424 2895     IF (.BASSA_RESTART EQLA 0) THEN BASS$STOP (BASSK_PROLOSSOR);
1425 2896
1426 2897     IF (.HIGHEST_LEVEL EQL 0)
1427 2898     THEN
1428 2899     BEGIN
1429 2900
1430 2901     *
1431 2902     Rather than doing an unwind to 0, search through the frames and patch
1432 2903     the return PC.
1433 2904
1434 2905     THIS_FMP = .FP;
1435 2906
1436 2907     DO
1437 2908     BEGIN
1438 2909     NEXT_FMP = .THIS_FMP;
1439 2910     THIS_FMP = .THIS_FMP [BSFSA_SAVED_FP];
1440 2911     END
1441 2912     UNTIL (.THIS_FMP EQLA .HIGHEST_FMP);
1442 2913
1443 2914     NEXT_FMP [BSFSA_SAVED_PC] = RESTART;
1444 2915     END;
1445 2916
1446 2917     RETURN (USER_HAND_CONT);
1447 2918     END;
```

```
1446 2917
1447 2918
1448 2919
1449 2920
1450 2921
1451 2922
1452 2923
1453 2924
1454 2925
1455 2926
1456 2927
1457 2928
1458 2929
1459 2930
1460 2931
1461 2932
1462 2933
1463 2934
1464 2935
1465 2936
1466 2937
1467 2938
1468 2939
1469 2940
1470 2941
1471 2942
1472 2943
1473 2944
1474 2945
1475 2946
1476 2947
1477 2948
1478 2949
1479 2950
1480 2951
1481 2952
1482 2953
1483 2954
1484 2955
1485 2956
1486 2957
1487 2958
1488 2959
1489 2960
1490 2961
1491 2962
1492 2963
1493 2964
1494 2965
1495 2966
1496 2967
1497 2968
1498 2969
1499 2970
1500 2971
1501 2972
1502 2973

[USER_ERR_GOBK] :
+
The condition handler ended with ON ERROR GO BACK. Revert but
continue to call BAS$$USER HAND. However, this frame is marked for
an immediate ON ERROR GO BACK in case we are in a GOSUB: we don't
want to call the user's error handler again.
-
BEGIN
  BAS$L_ERRFLG = 0;
  CASE .MAJOR_OR_MINOR FROM K_MINOR TO K_MAJOR OF
    SET
      [K_MINOR] :
        BSF$A_MINOR_STG [BSF$A_USER_HAND] = 1;
      [K_MAJOR] :
        BSF$A_MAJOR_STG [BSF$A_USER_HAND] = 1;
    TES;
  RETURN (USER_HAND_BACK);
END;

[USER_ERR_OEGZ] :
+
The condition handler ended with ON ERROR GOTO 0. Revert but
force system handling for this error.
-
BEGIN
  CASE .MAJOR_OR_MINOR FROM K_MINOR TO K_MAJOR OF
    SET
      [K_MINOR] :
        BSF$A_MINOR_STG [BSF$A_USER_HAND] = 0;
      [K_MAJOR] :
        BSF$A_MAJOR_STG [BSF$A_USER_HAND] = 0;
    TES;
  RETURN (USER_HAND_FAIL);
END;

[OTHERWISE] :
+
The condition handler ended with a RESUME with a line number.
Unwind to the current frame and restart at the indicated PC.
-
BEGIN
  BAS$L_ERRFLG = 0;
  BAS$A_RESTART = .ONER_RESULT;
  BAS$A_CH_CUR_LN = 0;
  BAS$L_GOTNG_BACK = 0;
  UNWIND_COUNT = .ACCUM_LEVEL;
  IF (.LEVEL EQL 0)
```



```
1503 2974 3 THEN
1504 2975 4 BEGIN
1505 2976 4
1506 2977 4 * Rather than doing an unwind to 0, search through the frames and
1507 2978 4 patch the return PC.
1508 2979 4
1509 2980 4 THIS_FMP = .FP;
1510 2981 4
1511 2982 4 DO
1512 2983 5 BEGIN
1513 2984 5 NEXT_FMP = .THIS_FMP;
1514 2985 5 THIS_FMP = .THIS_FMP [BSFSA_SAVED_FP];
1515 2986 5 END
1516 2987 4 UNTIL (.THIS_FMP EQLA .FMP);
1517 2988 4
1518 2989 4 NEXT_FMP [BSFSA_SAVED_PC] = RESTART;
1519 2990 4
1520 2991 4 END;
1521 2992 4
1522 2993 4 RETURN (USER_HAND_CONT);
1523 2994 4 END
1524 2995 4 TES;
1525 2996 4
1526 2997 1 END;
```

! of BAS\$\$USER_HAND

```
007C 00000 BAS$$USER HAND:
56 FC9A CF 9E 00002 .ORD Save R2,R3,R4,R5,R6 2632
55 00000000 EF 9E 00007 MOVAB ERR_SEVERITY, R6
50 66 9E 0000E MOVAB BASL_ERRFLG, R5
02 04 BC40 91 00011 CMPB ERR_SEVERITY, R0
06 13 00016 BEQL @ERR_CODE[R0], #2
04 BC40 95 00018 TSTB @ERR_CODE[R0]
02 12 0001C BNEQ 1$
65 D5 0001E 1$: TSTL BASL_ERRFLG
03 13 00020 2$: BEQL 3$
013C 31 00022 BRW 30$
65 01 D0 00025 3$: MOVL #1, BASL_ERRFLG
2C A5 D5 00028 TSTL BASL_GOING_BACK
0C AC C0 0002D BEQL 4$
54 11 00032 ADDL2 LEVEL, ACCUM_LEVEL
32 04 AC D1 00034 4$: BRB 8$
06 13 00038 CMPL ERR_CODE, #50
34 04 AC D1 0003A BEQL 5$
04 12 0003E CMPL ERR_CODE, #52
50 D4 00040 5$: BNEQ 6$
03 11 00042 CLRL R0
2C 50 01 D0 00044 6$: BRB 7$
A5 50 D0 00047 7$: MOVL #1, R0
52 08 AC D0 0004B MOVL R0, BASL_GOING_BACK
28 A5 FC A2 D0 0004F MOVL FMP, R2
52 DD 00054 MOVL -4(R2), BASA_CH_CUR_LN
PUSHL R2 2772
2773
```

FEED	CF	01	FB	00056	CALLS	#1, BAS\$LINE	
FC	AS	50	DD	0005B	MOVL	R0, BAS\$L_ERN	2774
		52	DD	0005F	PUSHL	R2	
FF20	CF	01	FB	00061	CALLS	#1, BAS\$MODULE	
F4	AS		A0	9E	MOVAB	1(R0), BAS\$T_ERN+4	2775
F0	AS		60	9B	MOVZBW	(MOD_NAME_ADDR), BAS\$T_ERN	2776
F2	AS	010E	8F	B0	MOVW	#270, BAS\$T_ERN+2	2778
F8	AS	04	AC	D0	MOVL	ERR_CODE, BAS\$L_ERR	2779
04	AS	0C	AC	D0	MOVL	LEVEL, HIGHEST_LEVEL	2780
08	AS		52	D0	MOVL	R2, HIGHEST_FMP	2781
0C	AS	0C	AC	D0	MOVL	LEVEL, ACCUM_LEVEL	2782
	50	08	AC	D0	MOVL	FMP, NON_GOSUB_FMP	2789
			52	D4	CLRL	SEARCH_DONE	2790
	51	00000000G	00	9E	MOVAB	BAS\$HANDLER, R1	2795
	51		60	D1	CMPL	(NON_GOSUB_FMP), R1	
			09	12	BNEQ	10\$	
	06	E5	A0	91	CMPS	-27(NON_GOSUB_FMP), #6	2798
			03	13	BEQL	10\$	
	52		01	D0	MOVL	#1, SEARCH_DONE	
	09		52	E8	BLBS	SEARCH_DONE, 11\$	2800
	50	0C	A0	D0	MOVL	12(NON_GOSUB_FMP), NON_GOSUB_FMP	2803
			37	13	BEQL	17\$	2805
	DF		52	E9	BLBC	SEARCH_DONE, 9\$	2816
001F	01	E5	A0	8F	CASEB	-27(NON_GOSUB_FMP), #1, #7	2818
002B	0012		0012		.WORD	13\$-12\$,-	
	002B		0012			13\$-12\$,-	
						13\$-12\$,-	
						14\$-12\$,-	
						13\$-12\$,-	
						15\$-12\$,-	
						15\$-12\$,-	
						15\$-12\$,-	
						15\$-12\$	
			19	11	BRB	15\$	2841
	53	F4	A0	D0	MOVL	-12(NON_GOSUB_FMP), BSF\$A_MAJOR_STG	2823
	51	7F	A3	D0	MOVL	127(BSF\$A_MAJOR_STG), USER_HAND_VAL	2824
	54		01	D0	MOVL	#1, MAJOR_OR_MINOR	2825
			0E	11	BRB	16\$	2818
	52	F0	A0	D0	MOVL	-16(NON_GOSUB_FMP), BSF\$A_MINOR_STG	2830
	51	7F	A2	D0	MOVL	127(BSF\$A_MINOR_STG), USER_HAND_VAL	2831
			54	D4	CLRL	MAJOR_OR_MINOR	2832
			02	11	BRB	16\$	2818
			51	D4	CLRL	USER_HAND_VAL	2837
			51	D5	TSTL	USER_HAND_VAL	2844
			7C	13	BEQL	30\$	
	01		51	D1	CMPL	USER_HAND_VAL, #1	2852
			04	12	BNEQ	18\$	
			65	D4	CLRL	BAS\$L_ERRFLG	2862
			5A	11	BRB	25\$	2863
			03	BB	PUSHR	#*M<R0,R1>	2871
00000000G	00		02	FB	CALLS	#2, BAS\$INIT_ONERR	
			50	D5	TSTL	ONERR_RESULT	2876
			34	12	BNEQ	21\$	
			65	D4	CLRL	BAS\$L_ERRFLG	2883
	30	AS	AS	D0	MOVL	BAS\$A_CH_CUR_LN, BAS\$A_RESTART	2884
			AS	7C	CLRL	BAS\$A_CH_CUR_LN	2885
	10	AS	AS	D0	MOVL	HIGHEST_LEVEL, UNWIND_COUNT	2887
		30	AS	D5	TSTL	BAS\$A_RESTART	2894

			09	12	0010D	BNEQ	198		
	7E	00G	8F	9A	0010F	MOVZBL	#BAS\$K_PROLOSSOR, -(SP)		
FD9C	CF		01	FB	00113	CALLS	#1, BAS\$\$STOP		
		04	A5	D5	00118	TSTL	HIGHEST_LEVEL	2896	
			71	12	0011B	BNEQ	348		
	51		5D	D0	0011D	MOVL	FP, THIS_FMP	2903	
	50		51	D0	00120	MOVL	THIS_FMP, NEXT_FMP	2907	
	51		A1	D0	00123	MOVL	12(THIS_FMP), THIS_FMP	2908	
08	A5	0C	51	D1	00127	CMPL	THIS_FMP, HIGHEST_FMP	2910	
			F3	12	0012B	BNEQ	208		
			59	11	0012D	BRB	338	2912	
	01		50	D1	0012F	CMPL	ONER_RESULT, #1	2918	
			18	12	00132	BNEQ	268		
			65	D4	00134	CLRL	BAS\$L_ERRFLG	2926	
01	00		54	CF	00136	CASEL	MAJOR_OR_MINOR, #0, #1	2928	
	000A	0004			0013A	.WORD	238-228, =		
							248-228		
7F	A2		01	D0	0013E	MOVL	#1, 127(BSF\$A_MINOR_STG)	2932	
			04	11	00142	BRB	258		
7F	A3		01	D0	00144	MOVL	#1, 127(BSF\$A_MAJOR_STG)	2935	
	50		01	D0	00148	MOVL	#1, R0	2938	
				04	0014B	RET			
	02		50	D1	0014C	CMPL	ONER_RESULT, #2	2941	
			14	12	0014F	BNEQ	318		
01	00		54	CF	00151	CASEL	MAJOR_OR_MINOR, #0, #1	2948	
	0009	0004			00155	.WORD	288-278, =		
							298-278		
		7F	A2	D4	00159	CLRL	127(BSF\$A_MINOR_STG)	2952	
			03	11	0015C	BRB	308		
		7F	A3	D4	0015E	CLRL	127(BSF\$A_MAJOR_STG)	2955	
	50		02	D0	00161	MOVL	#2, R0	2958	
				04	00164	RET			
			65	D4	00165	CLRL	BAS\$L_ERRFLG	2967	
30	A5		50	D0	00167	MOVL	ONER_RESULT, BAS\$A_RESTART	2968	
		28	A5	7C	0016B	CLRQ	BAS\$A_CH_CUR_LN	2969	
10	A5	0C	A5	D0	0016E	MOVL	ACCUM_LEVEL, UNWIND_COUNT	2971	
		0C	AC	D5	00173	TSTL	LEVEL	2973	
			16	12	00176	BNEQ	348		
	51		5D	D0	00178	MOVL	FP, THIS_FMP	2980	
	50		51	D0	0017B	MOVL	THIS_FMP, NEXT_FMP	2984	
	51	0C	A1	D0	0017E	MOVL	12(THIS_FMP), THIS_FMP	2985	
08	AC		51	D1	00182	CMPL	THIS_FMP, FMP	2987	
			F3	12	00186	BNEQ	328		
10	A0	FE44	CF	9E	00188	MOVAB	RESTART, 16(NEXT_FMP)	2989	
			50	D4	0018E	CLRL	R0	2993	
				04	00190	RET		2997	

; Routine Size: 401 bytes. Routine Base: _BAS\$CODE + 0360

; 1527 2998 1

```
1529 2999 1 GLOBAL ROUTINE BASSRESUME (      ! end of error handler
1530 3000 1     NEW_PC                          ! where to restart
1531 3001 1     ) =
1532 3002 1
1533 3003 1
1534 3004 1 ++
1535 3005 1 FUNCTIONAL DESCRIPTION:
1536 3006 1
1537 3007 1     Resume execution from an error handler. The compiled code
1538 3008 1     calls RESUME passing the address of the location at which
1539 3009 1     to continue execution. We must be in an error handler.
1540 3010 1     The stack is cut back to the call to BASSINIT_ONER which
1541 3011 1     is in BASS$USER_HAND and the RET at the end of this routine
1542 3012 1     actually returns from BASSINIT_ONER. This is similar to
1543 3013 1     GOSUB processing. To simplify the restoring of registers
1544 3014 1     BASSINIT_ONER saves them all, so the return to BASS$USER_HAND
1545 3015 1     restores them.
1546 3016 1
1547 3017 1     If necessary, GOSUB frames are removed looking for the
1548 3018 1     condition handling frame, but if another type of frame
1549 3019 1     is encountered we have an error. If there is no error
1550 3020 1     pending then the RESUME is turned into a GOTO, for
1551 3021 1     compatibility with BASIC-PLUS.
1552 3022 1
1553 3023 1 FORMAL PARAMETERS:
1554 3024 1     NEW_PC.ra.v    The location at which to continue execution.
1555 3025 1
1556 3026 1 IMPLICIT INPUTS:
1557 3027 1
1558 3028 1     NONE
1559 3029 1
1560 3030 1 IMPLICIT OUTPUTS:
1561 3031 1
1562 3032 1     NONE
1563 3033 1
1564 3034 1 ROUTINE VALUE:
1565 3035 1
1566 3036 1     The resume PC. This is returned to the caller of BASSINIT_ONER,
1567 3037 1     which is presumed to be BASS$USER_HAND.
1568 3038 1
1569 3039 1 COMPLETION CODES:
1570 3040 1
1571 3041 1     NONE
1572 3042 1
1573 3043 1 SIDE EFFECTS:
1574 3044 1
1575 3045 1     May cut back the stack, thus not returning to the caller.
1576 3046 1     If it does return, it is not to the call site but to the
1577 3047 1     location specified in the parameter.
1578 3048 1
1579 3049 1 --
1580 3050 1
1581 3051 2 BEGIN
1582 3052 2
1583 3053 2 BUILTIN
1584 3054 2     FP;
1585 3055 2
```



```
1586 3056 LOCAL
1587 3057 FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);
1588 3058
1589 3059
1590 3060
1591 3061
1592 3062
1593 3063
1594 3064
1595 3065
1596 3066
1597 3067
1598 3068
1599 3069
1600 3070
1601 3071
1602 3072
1603 3073
1604 3074
1605 3075
1606 3076
1607 3077
1608 3078
1609 3079
1610 3080
1611 3081
1612 3082
1613 3083
1614 3084
1615 3085
1616 3086
1617 3087
1618 3088
1619 3089
1620 3090
1621 3091
1622 3092
1623 3093
1624 3094
1625 3095
1626 3096
1627 3097
1628 3098
1629 3099
1630 3100
1631 3101
1632 3102
1633 3103
1634 3104
1635 3105
1636 3106
1637 3107
1638 3108
1639 3109
1640 3110
1641 3111
1642 3112

LOCAL
FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);

+
If there is no error being processed, stuff the parameter into the
return address and return, thus turning the RESUME statement into
a GOTO.
-

IF (.BAS$L_ERRFLG EQL 0)
THEN
BEGIN
FMP = .FP;
FMP [BSF$A_SAVED_PC] = .NEW_PC;
RETURN (0);
END;
! the value will be ignored

+
Dig back through GOSUB frames to find the condition handling frame.
-
FMP = .FP;
FMP = .FMP [BSF$A_SAVED_FP];
WHILE (.FMP [BSF$B_PROC_CODE] EQL BSF$K_PROC_GOSB) DO
BEGIN
FMP = .FMP [BSF$A_SAVED_FP];
IF (.FMP [BSF$A_HANDLER] NEQA BAS$HANDLER)
THEN
+
The previous frame is not a BASIC frame. This means that the user
began processing an error, called a non-BASIC routine which called
a BASIC routine which tried to dismiss the error. Disallow this
kind of poorly-structured code.
-
BAS$$SIGNAL (BAS$K_RESNO_ERR);

+
Deallocate any heap storage that may be held by this frame.
-
BAS$$UNWIND (.FMP);
END;

+
We have finished cutting back the GOSUB frames. Now be sure we are
in the condition handler.
-

IF (.FMP [BSF$B_PROC_CODE] NEQ BSF$K_PROC_ONER)
THEN
+
We are not. This can happen if the user begins processing an error,
then calls another routine which tries to dismiss the error.
Disallow this, also.
-
BAS$$SIGNAL (BAS$K_RESNO_ERR);
```

```
1643 3113 1
1644 3114 2
1645 3115 3
1646 3116 4
1647 3117 5
1648 3118 6
1649 3119 7
1650 3120 8
1651 3121 9
1652 3122 10
1653 3123 11
1654 3124 12
1655 3125 13
1656 3126 14
1657 3127 15
1658 3128 16
1659 3129 17
```

↑ We have reached the condition handling frame. By stuffing the pointer to this frame into FP we effectively cut back the stack, since this routine's RET will then return from the condition handler. Note that we are not restoring any registers; we depend on the fact that BAS\$INIT_ONER saves them all, so the RET we are about to do will restore registers for BAS\$\$USER_HAND.

BAS\$\$UNWIND (.FMP);
FP = .FMP;

↑ By returning a number whose unsigned value is greater than 2 we indicate to BAS\$\$USER_HAND that the user has written RESUME with a line number.

RETURN (.NEW_PC);
END;

! of BAS\$RESUME

			000C 00000	.ENTRY	BAS\$RESUME, Save R2,R3	2999
	53	00000000G	00 9E 00002	MOVAB	BAS\$\$UNWIND, R3	
		00000000'	EF 05 00009	TSTL	BAS\$L_ERRFLG	3065
			0A 12 0000F	BNEQ	1\$	
	52		5D 00 00011	MOVL	FP, FMP	3068
10	A2	04	AC 00 00014	MOVL	NEW_PC, 16(FMP)	3069
			49 11 00019	BRB	6\$	3070
	52		5D 00 0001B	MOVL	FP, FMP	3076
	52	0C	A2 00 0001E	MOVL	12(FMP), FMP	3077
	06	E5	A2 91 00022	CMPB	-27(FMP), #6	3079
			20 12 00026	BNEQ	4\$	
	52	0C	A2 00 00028	MOVL	12(FMP), FMP	3081
	50	00000000G	00 9E 0002C	MOVAB	BAS\$HANDLER, R0	3083
	50		62 01 00033	CPL	(FMP), R0	
			09 13 00036	BEQ	3\$	
	7E	00G	8F 9A 00038	MOVZBL	#BAS\$K_RESNO_ERR, -(SP)	3091
FCCE	CF		01 FB 0003C	CALLS	#1, BAS\$\$SIGNAL	
			52 DD 00041	PUSHL	FMP	3096
	63		01 FB 00043	CALLS	#1, BAS\$\$UNWIND	
			DA 11 00046	BRB	2\$	3079
	07	E5	A2 91 00048	CMPB	-27(FMP), #7	3104
			09 13 0004C	BEQ	5\$	
	7E	00G	8F 9A 0004E	MOVZBL	#BAS\$K_RESNO_ERR, -(SP)	3111
FCB8	CF		01 FB 00052	CALLS	#1, BAS\$\$SIGNAL	
			52 DD 00057	PUSHL	FMP	3121
	63		01 FB 00059	CALLS	#1, BAS\$\$UNWIND	
	50		52 00 0005C	MOVL	FMP, FP	3122
	50	04	AC 00 0005F	MOVL	NEW_PC, R0	3128
			04 00063	RET		
			50 04 00064	CLRL	R0	3129
			04 00066	RET		

; Routine Size: 103 bytes. Routine Base: _BAS\$CODE + 04F1

BASERROR
1-074

: 1660

3130 1

L 10
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

VAX-11 B11ss-32 V4.0-742
[BASRTL.SRC]BASERROR.B32;1

Page 47
(16)

```
1662 3131 1 GLOBAL ROUTINE BASS$RESUME_2 = ! Resume with no line number
1663 3132 1
1664 3133 1
1665 3134 1 ++
1666 3135 1 FUNCTIONAL DESCRIPTION:
1667 3136 1 Resume execution from an error handler. The compiled code
1668 3137 1 calls RESUME_2 to indicate that the statement in which the
1669 3138 1 error occurred is to be restarted or continued. (Which
1670 3139 1 depends on which error is in progress.)
1671 3140 1 The stack is cut back to the call to BASS$INIT_ONER which
1672 3141 1 is in BASS$USER_HAND and the RET at the end of this routine
1673 3142 1 actually returns from BASS$INIT_ONER. This is similar to
1674 3143 1 GOSUB processing. To simplify the restoring of registers
1675 3144 1 BASS$INIT_ONER saves them all, so the return to BASS$USER_HAND
1676 3145 1 restores them.
1677 3146 1
1678 3147 1 If necessary, GOSUB frames are removed looking for the
1679 3148 1 condition handling frame, but if another type of frame
1680 3149 1 is encountered we have an error.
1681 3150 1
1682 3151 1 FORMAL PARAMETERS:
1683 3152 1
1684 3153 1 NONE
1685 3154 1
1686 3155 1 IMPLICIT INPUTS:
1687 3156 1
1688 3157 1 NONE
1689 3158 1
1690 3159 1 IMPLICIT OUTPUTS:
1691 3160 1
1692 3161 1 NONE
1693 3162 1
1694 3163 1 ROUTINE VALUE:
1695 3164 1
1696 3165 1 USER_ERR_RSUM2, to indicate to BASS$USER_HAND that the user did
1697 3166 1 a RESUME with no line number.
1698 3167 1
1699 3168 1 COMPLETION CODES:
1700 3169 1
1701 3170 1 NONE
1702 3171 1
1703 3172 1 SIDE EFFECTS:
1704 3173 1
1705 3174 1 Cuts back the stack, thus not returning to the caller.
1706 3175 1
1707 3176 1 --
1708 3177 1
1709 3178 2 BEGIN
1710 3179 2
1711 3180 2 BUILTIN
1712 3181 2 FP;
1713 3182 2
1714 3183 2 LOCAL
1715 3184 2 FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);
1716 3185 2
1717 3186 2 ++
1718 3187 2 ! If there is no error being processed, the RESUME statement without
```



```
1719 3188 2 | a line number is invalid.
1720 3189 2 |
1721 3190 2 |
1722 3191 2 | IF (.BAS$L_ERRFLG EQL 0) THEN BAS$$SIGNAL (BAS$K_RESNO_ERR);
1723 3192 2 |
1724 3193 2 | +
1725 3194 2 | If we are not in the same program unit as the source of the error,
1726 3195 2 | we have an error. This is done for compatability with the PDP-11.
1727 3196 2 |
1728 3197 2 | FMP = .FP;
1729 3198 2 | FMP = .FMP [BSF$A_SAVED_FP];
1730 3199 2 |
1731 3200 2 | IF ((BAS$$MODULE (.FMP) + 1) NEQA .BAS$T_ERN [DSC$A_POINTER]) THEN BAS$$STOP (BAS$K_ILLRESSUB);
1732 3201 2 |
1733 3202 2 | +
1734 3203 2 | Dig back through GOSUB frames to find the condition handling frame.
1735 3204 2 |
1736 3205 2 |
1737 3206 2 | WHILE (.FMP [BSF$B_PROC_CODE] EQL BSF$K_PROC_GOSB) DO
1738 3207 2 | BEGIN
1739 3208 2 | FMP = .FMP [BSF$A_SAVED_FP];
1740 3209 2 |
1741 3210 2 | IF (.FMP [BSF$A_HANDLER] NEQA BAS$HANDLER)
1742 3211 2 | THEN
1743 3212 2 | +
1744 3213 2 | The previous frame is not a BASIC frame. This means that the user
1745 3214 2 | began processing an error, called a non-BASIC routine which called
1746 3215 2 | a BASIC routine which tried to dismiss the error. Disallow this
1747 3216 2 | kind of poorly-structured code.
1748 3217 2 |
1749 3218 2 | BAS$$SIGNAL (BAS$K_RESNO_ERR);
1750 3219 2 |
1751 3220 2 | +
1752 3221 2 | Deallocate any heap storage that may be held by this frame.
1753 3222 2 |
1754 3223 2 | BAS$$UNWIND (.FMP);
1755 3224 2 | END;
1756 3225 2 |
1757 3226 2 | +
1758 3227 2 | We have finished cutting back the GOSUB frames. Now be sure we are
1759 3228 2 | in the condition handler.
1760 3229 2 |
1761 3230 2 |
1762 3231 2 | IF (.FMP [BSF$B_PROC_CODE] NEQ BSF$K_PROC_ONER)
1763 3232 2 | THEN
1764 3233 2 | +
1765 3234 2 | We are not. This can happen if the user begins processing an error,
1766 3235 2 | then calls another routine which tries to dismiss the error.
1767 3236 2 | Disallow this, also.
1768 3237 2 |
1769 3238 2 | BAS$$SIGNAL (BAS$K_ILLRESSUB);
1770 3239 2 |
1771 3240 2 | +
1772 3241 2 | We have reached the condition handling frame. By stuffing the
1773 3242 2 | pointer to this frame into FP we effectively cut back the stack,
1774 3243 2 | since this routine's RET will then return from the condition
1775 3244 2 | handler. Note that we are not restoring any registers; we depend
```

```
1776 3245 2 on the fact that BASSINIT ONER saves them all, so the RET we are
1777 3246 about to do will restore registers for BASS$USER_HAND.
1778 3247
1779 3248 BASS$UNWIND (.FMP);
1780 3249 FP = .FMP;
1781 3250
1782 3251 + Indicate to BASS$USER_HAND that the user has written a RESUME
1783 3252 with no line number.
1784 3253
1785 3254 RETURN (USER_ERR_RSUMZ);
1786 3255 END;
```

! of BASS\$RESUME_Z

54	00000000G	00	9E	000002	.ENTRY	BASS\$RESUME_Z, Save R2,R3,R4	3131
53	FC9B	CF	9E	000009	MOVAB	BASS\$UNWIND, R4	
	00000000'	EF	D5	00000E	MOVAB	BASS\$SIGNAL, R3	
		07	12	000014	TSTL	BASSL_ERRFLG	3191
7E	00G	8F	9A	000016	BNEQ	1\$	
63		01	FB	00001A	MOVZBL	#BASSK_RESNO_ERR, -(SP)	
52		5D	D0	00001D	CALLS	#1, BASS\$SIGNAL	
52	0C	A2	D0	000020	MOVL	FP, FMP	3197
		52	DD	000024	MOVL	12(FMP), FMP	3198
00E6	C3	01	FB	000026	PUSHL	FMP	3200
		50	D6	00002B	CALLS	#1, BASS\$MODULE	
00000000'	EF	50	D1	00002D	INCL	R0	
		08	13	000034	CMPL	R0, BASST_ERN+4	
7E	00G	8F	9A	000036	BEQL	2\$	
14	A3	01	FB	00003A	MOVZBL	#BASSK_ILRESSUB, -(SP)	
06	E5	A2	91	00003E	CALLS	#1, BASS\$STOP	
		1E	12	000042	CMPL	-27(FMP), #6	3206
52	0C	A2	D0	000044	BNEQ	4\$	
50	00000000G	00	9E	000048	MOVL	12(FMP), FMP	3208
50		62	D1	00004F	MOVAB	BASS\$HANDLER, R0	3210
		07	13	000052	CMPL	(FMP), R0	
7E	00G	8F	9A	000054	BEQL	3\$	
63		01	FB	000058	MOVZBL	#BASSK_RESNO_ERR, -(SP)	3218
		52	DD	00005B	CALLS	#1, BASS\$SIGNAL	
64		01	FB	00005D	PUSHL	FMP	3223
		DC	11	000060	CALLS	#1, BASS\$UNWIND	
07	E5	A2	91	000062	BRB	2\$	3206
		07	13	000066	CMPL	-27(FMP), #7	3231
7E	00G	8F	9A	000068	BEQL	5\$	
63		01	FB	00006C	MOVZBL	#BASSK_ILRESSUB, -(SP)	3238
		52	DD	00006F	CALLS	#1, BASS\$SIGNAL	
64		01	FB	000071	PUSHL	FMP	3248
5D		52	D0	000074	CALLS	#1, BASS\$UNWIND	
		50	D4	000077	MOVL	FMP, FP	3249
		04	000079	CLRL	R0		3254
				RET			3255

; Routine Size: 122 bytes, Routine Base: _BAS\$CODE + 0558

; 1787 3256 1

```
1789 3257 1 GLOBAL ROUTINE BASSON_ERR_Z =
1790 3258 1
1791 3259 1
1792 3260 1 **
1793 3261 1 FUNCTIONAL DESCRIPTION:
1794 3262 1
1795 3263 1 The BASIC statement ON ERROR GOTO 0 is compiled as
1796 3264 1
1797 3265 1 CALLS #0,BASSON_ERR_Z
1798 3266 1 CLRL BASSA_USER_HAND(Rn)
1799 3267 1
1800 3268 1 Thus, the job of BASSON_ERR_Z is to return if there is no
1801 3269 1 error in progress, and to provide system error processing
1802 3270 1 if there is an error in progress. This latter function is
1803 3271 1 done by cutting back the stack (like RESUME) and returning
1804 3272 1 to BASS$USER_HAND indicating that system error handling
1805 3273 1 is required.
1806 3274 1
1807 3275 1 FORMAL PARAMETERS:
1808 3276 1
1809 3277 1 NONE
1810 3278 1
1811 3279 1 IMPLICIT INPUTS:
1812 3280 1
1813 3281 1 BASSL_ERRFLG
1814 3282 1
1815 3283 1 IMPLICIT OUTPUTS:
1816 3284 1
1817 3285 1 NONE
1818 3286 1
1819 3287 1 ROUTINE VALUE:
1820 3288 1
1821 3289 1 Either no value, or USER_ERR_OEGZ, to indicate to BASS$USER_HAND
1822 3290 1 that the user did an ON ERROR GOTO 0 in his error handler.
1823 3291 1
1824 3292 1 COMPLETION CODES:
1825 3293 1
1826 3294 1 NONE
1827 3295 1
1828 3296 1 SIDE EFFECTS:
1829 3297 1
1830 3298 1 May cut back the stack, thus not returning to the caller.
1831 3299 1
1832 3300 1 --
1833 3301 1 BEGIN
1834 3302 1
1835 3303 1 BUILTIN
1836 3304 1 FP:
1837 3305 1
1838 3306 1 LOCAL
1839 3307 1 FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);
1840 3308 1
1841 3309 1
1842 3310 1 If there is no error being processed, just return. The value will
1843 3311 1 be ignored.
1844 3312 1
1845 3313 2
```

```
1846 3314 IF (.BASL_ERRFLG EQL 0) THEN RETURN (0);
1847 3315
1848 3316
1849 3317
1850 3318
1851 3319
1852 3320
1853 3321
1854 3322
1855 3323
1856 3324
1857 3325
1858 3326
1859 3327
1860 3328
1861 3329
1862 3330
1863 3331
1864 3332
1865 3333
1866 3334
1867 3335
1868 3336
1869 3337
1870 3338
1871 3339
1872 3340
1873 3341
1874 3342
1875 3343
1876 3344
1877 3345
1878 3346
1879 3347
1880 3348
1881 3349
1882 3350
1883 3351
1884 3352
1885 3353
1886 3354
1887 3355
1888 3356
1889 3357
1890 3358
1891 3359
1892 3360
1893 3361
1894 3362
1895 3363
1896 3364
1897 3365
1898 3366
1899 3367
1900 3368
1901 3369
1902 3370

    IF (.BASL_ERRFLG EQL 0) THEN RETURN (0);

    Dig back through GOSUB frames to find the condition handling frame.
    FMP = .FP;
    FMP = .FMP [BSFSA_SAVED_FP];

    WHILE (.FMP [BSFSB_PROC_CODE] EQL BSFSK_PROC_GOSB) DO
        BEGIN
            FMP = .FMP [BSFSA_SAVED_FP];

            IF (.FMP [BSFSA_HANDLER] NEQA BAS$HANDLER)
            THEN
                The previous frame is not a BASIC frame. This means that the user
                began processing an error, called a non-BASIC routine which called
                a BASIC routine which tried to specify system error handling.
                Disallow this kind of poorly-structured code.

                BAS$$SIGNAL (BAS$K_RESNO_ERR);

            Deallocate any heap storage that may be held by this frame.

            BAS$$UNWIND (.FMP);
            END;

            We have finished cutting back the GOSUB frames. Now be sure we are
            in the condition handler.

            IF (.FMP [BSFSB_PROC_CODE] NEQ BSFSK_PROC_ONER)
            THEN
                We are not. This can happen if the user begins processing an error,
                then calls another routine which tries to specify system error
                handling. Disallow this, also.

                BAS$$SIGNAL (BAS$K_RESNO_ERR);

            We have reached the condition handling frame. By stuffing the
            pointer to this frame into FP we effectively cut back the stack,
            since this routine's RET will then return from the condition
            handler. Note that we are not restoring any registers; we depend
            on the fact that BAS$INIT_ONER saves them all, so the RET we are
            about to do will restore registers for BAS$$USER_HAND.

            BAS$$UNWIND (.FMP);
            FP = .FMP;

            Indicate to BAS$$USER_HAND that the user has written ON ERROR GOTO 0.

            RETURN (USER_ERR_OEGZ);
            END;
```

! of BAS\$ON_ERR_2

			000C 00000	.ENTRY	BASSON ERR 2, Save R2,R3	.. 3257
53	00000000G	00	9E 00002	MOVAB	BASSUNWIND, R3	.. 3314
	00000000'	EF	D5 00009	TSTL	BASSL_ERRFLG	.. 3319
		48	13 0000F	BEQL	5\$.. 3320
52		5D	D0 00011	MOVL	FP, FMP	.. 3322
52	0C	A2	D0 00014	MOVL	12(FMP), FMP	.. 3324
06	E5	A2	91 00018	1\$: CMPB	-27(FMP), #6	.. 3326
		20	12 0001C	BNEQ	3\$.. 3334
52	0C	A2	D0 0001E	MOVL	12(FMP), FMP	.. 3339
50	00000000G	00	9E 00022	MOVAB	BASSHANDLER, R0	.. 3347
50		62	D1 00029	CMPL	(FMP), R0	.. 3354
		09	13 0002C	BEQL	2\$.. 3364
7E	00G	8F	9A 0002E	MOVZBL	#BASSK RESNO ERR, -(SP)	.. 3365
FBF7	CF	01	FB 00032	CALLS	#1, BASS\$SIGNAL	.. 3369
		52	DD 00037	2\$: PUSHL	FMP	.. 3370
63		01	FB 00039	CALLS	#1, BASS\$UNWIND	.. 3370
		DA	11 0003C	BRB	1\$.. 3370
07	E5	A2	91 0003E	3\$: CMPB	-27(FMP), #7	.. 3370
		09	13 00042	BEQL	4\$.. 3370
7E	00G	8F	9A 00044	MOVZBL	#BASSK RESNO ERR, -(SP)	.. 3370
FBE1	CF	01	FB 00048	CALLS	#1, BASS\$SIGNAL	.. 3370
		52	DD 0004D	4\$: PUSHL	FMP	.. 3370
63		01	FB 0004F	CALLS	#1, BASS\$UNWIND	.. 3370
5D		52	D0 00052	MOVL	FMP, FP	.. 3370
50		02	D0 00055	MOVL	#2, R0	.. 3370
			04 00058	RET		.. 3370
		50	D4 00059	5\$: CLRL	R0	.. 3370
			04 0005B	RET		.. 3370

; Routine Size: 92 bytes, Routine Base: _BASS\$CODE + 05D2

; 1903 3371 1

```
1905 3372 1 GLOBAL ROUTINE BASSON_ERR_BK =
1906 3373 1                                     ! ON ERROR GO BACK
1907 3374 1
1908 3375 1 **
1909 3376 1 FUNCTIONAL DESCRIPTION:
1910 3377 1     The BASIC statement ON ERROR GO BACK is compiled as
1911 3378 1
1912 3379 1     CALLS  #0,BASSON_ERR_BK
1913 3380 1     MOVZBL #1,BASSA_USER_HAND(Rn)
1914 3381 1
1915 3382 1     Thus, the job of BASSON_ERR_BK is to return if there is no
1916 3383 1     error in progress, and to permit the caller of the BASIC
1917 3384 1     procedure to try to process the error if there is an error
1918 3385 1     in progress. This latter function is done by cutting back
1919 3386 1     the stack (like RESUME) and returning to BASS$USER_HAND with
1920 3387 1     a USER_ERR_GOBK, indicating that it is to revert.
1921 3388 1
1922 3389 1 FORMAL PARAMETERS:
1923 3390 1
1924 3391 1     NONE
1925 3392 1
1926 3393 1 IMPLICIT INPUTS:
1927 3394 1
1928 3395 1     BASSL_ERRFLG
1929 3396 1
1930 3397 1 IMPLICIT OUTPUTS:
1931 3398 1
1932 3399 1     NONE
1933 3400 1
1934 3401 1 ROUTINE VALUE:
1935 3402 1
1936 3403 1     Either no value, or USER_ERR_GOBK.
1937 3404 1
1938 3405 1 COMPLETION CODES:
1939 3406 1
1940 3407 1     NONE
1941 3408 1
1942 3409 1 SIDE EFFECTS:
1943 3410 1
1944 3411 1     May cut back the stack, thus not returning to the caller.
1945 3412 1
1946 3413 1 --
1947 3414 1
1948 3415 1 BEGIN
1949 3416 1
1950 3417 1 BUILTIN
1951 3418 1     FP;
1952 3419 1
1953 3420 1 LOCAL
1954 3421 1     FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD);
1955 3422 1
1956 3423 1
1957 3424 1 If there is no error being processed, just return. The value will
1958 3425 1 be ignored.
1959 3426 1
1960 3427 1
1961 3428 1 IF (.BASSL_ERRFLG EQL 0) THEN RETURN (0);
```

```
1962 3429
1963 3430
1964 3431
1965 3432
1966 3433
1967 3434
1968 3435
1969 3436
1970 3437
1971 3438
1972 3439
1973 3440
1974 3441
1975 3442
1976 3443
1977 3444
1978 3445
1979 3446
1980 3447
1981 3448
1982 3449
1983 3450
1984 3451
1985 3452
1986 3453
1987 3454
1988 3455
1989 3456
1990 3457
1991 3458
1992 3459
1993 3460
1994 3461
1995 3462
1996 3463
1997 3464
1998 3465
1999 3466
2000 3467
2001 3468
2002 3469
2003 3470
2004 3471
2005 3472
2006 3473
2007 3474
2008 3475
2009 3476
2010 3477
2011 3478
2012 3479
2013 3480
2014 3481
2015 3482
2016 3483
2017 3484
2018 3485

+ Dig back through GOSUB frames to find the condition handling frame.
-
  FMP = .FMP;
  FMP = .FMP [BSF$A_SAVED_FP];

  WHILE (.FMP [BSF$B_PROC_CODE] EQL BSF$K_PROC_GOSB) DO
  BEGIN
    FMP = .FMP [BSF$A_SAVED_FP];

    IF (.FMP [BSF$A_HANDLER] NEQA BAS$HANDLER)
    THEN
+ The previous frame is not a BASIC frame. This means that the user
- began processing an error, called a non-BASIC routine which called
  a BASIC routine which tried to allow the caller of the routine that
  got the error to handle the error. Disallow this kind of
  poorly-structured code.
-
    BAS$$SIGNAL (BAS$K_RESNO_ERR);

+ Deallocate any heap storage that may be held by this frame.
-
    BAS$$UNWIND (.FMP);
    END;

+ We have finished cutting back the GOSUB frames. Now be sure we are
- in the condition handler.
  IF (.FMP [BSF$B_PROC_CODE] NEQ BSF$K_PROC_ONER)
  THEN
+ We are not. This can happen if the user begins processing an error,
- then calls another routine which tries to specify system error
  handling. Disallow this, also.
-
    BAS$$SIGNAL (BAS$K_IMPERRHAN);

+ We have reached the condition handling frame. By stuffing the
- pointer to this frame into FP we effectively cut back the stack,
  since this routine's RET will then return from the condition
  handler. Note that we are not restoring any registers; we depend
  on the fact that BAS$INIT_ONER saves them all, so the RET we are
  about to do will restore registers for BAS$$USER_HAND.
-
    BAS$$UNWIND (.FMP);
    FP = .FMP;

+ Indicate to BAS$$USER_HAND that the user has written ON ERROR GO BACK.
-
    RETURN (USER_ERR_GOBK);
  END;

! of BAS$ON_ERR_BK
```

			000C 00000	.ENTRY	BASSON_ERR_BK, Save R2,R3	3372
53	00000000G	00	9E 00002	MOVAB	BASS\$UNWIND, R3	
	00000000'	EF	D5 00009	TSTL	BASSL_ERRFLG	3428
		48	13 0000F	BEQL	5\$	
52		5D	D0 00011	MOVL	FP, FMP	3433
52	OC	A2	D0 00014	MOVL	12(FMP), FMP	3434
06	E5	A2	91 00018 1\$:	CMPL	-27(FMP), #6	3436
		20	12 0001C	BNEQ	3\$	
52	OC	A2	D0 0001E	MOVL	12(FMP), FMP	3438
50	00000000G	00	9E 00022	MOVAB	BASS\$HANDLER, R0	3440
50		62	D1 00029	CMPL	(FMP), R0	
		09	13 0002C	BEQL	2\$	
7E	00G	8F	9A 0002E	MOVZBL	#BASSK_RESNO_ERR, -(SP)	3449
FB9B	CF	01	FB 00032	CALLS	#1, BASS\$SIGNAL	
		52	DD 00037 2\$:	PUSHL	FMP	3454
63		01	FB 00039	CALLS	#1, BASS\$UNWIND	
		DA	11 0003C	BRB	1\$	3436
07	E5	A2	91 0003E 3\$:	CMPL	-27(FMP), #7	3462
		09	13 00042	BEQL	4\$	
7E	00G	8F	9A 00044	MOVZBL	#BASSK_IMPERRHAN, -(SP)	3469
FB85	CF	01	FB 00048	CALLS	#1, BASS\$SIGNAL	
		52	DD 0004D 4\$:	PUSHL	FMP	3479
63		01	FB 0004F	CALLS	#1, BASS\$UNWIND	
5D		52	D0 00052	MOVL	FMP, FMP	3480
50		01	D0 00055	MOVL	#1, R0	3484
			04 00058	RET		
		50	D4 00059 5\$:	CLRL	R0	3485
			04 0005B	RET		

; Routine Size: 92 bytes, Routine Base: _BASS\$CODE + 062E

; 2019 3486 1


```
2021 3487 1 GLOBAL ROUTINE BASS$HANDLER (
2022 3488 1     SIGNAL_ARGS,
2023 3489 1     MECHANISM_ARGS
2024 3490 1 ) =
2025 3491 1
2026 3492 1
2027 3493 1 ++
2028 3494 1 FUNCTIONAL DESCRIPTION:
2029 3495 1
2030 3496 1     Handle an exception from within a BASIC-PLUS-2 routine.
2031 3497 1     Note that the real entry point, BASS$HANDLER, is a location in
2032 3498 1     the sharable library's vector (or in a small module if this code
2033 3499 1     is not shared) so that a frame can be tested for being a BASIC
2034 3500 1     frame by testing for BASS$HANDLER in 0(FP).
2035 3501 1
2036 3502 1 FORMAL PARAMETERS:
2037 3503 1
2038 3504 1     SIGNAL_ARGS    A vector of longwords which indicate the nature
2039 3505 1     MECHANISM_ARGS A vector of longwords that indicate the state
2040 3506 1                    of the process at the time of the signal.
2041 3507 1
2042 3508 1 IMPLICIT INPUTS:
2043 3509 1
2044 3510 1     The information in the frames of the BASIC-PLUS-2 routines
2045 3511 1     in and before the one which encountered the error.
2046 3512 1
2047 3513 1 IMPLICIT OUTPUTS:
2048 3514 1
2049 3515 1     NONE
2050 3516 1
2051 3517 1 ROUTINE VALUE:
2052 3518 1
2053 3519 1     An indication to the VAX/VMS CHF of whether or not to revert.
2054 3520 1
2055 3521 1 COMPLETION CODES:
2056 3522 1
2057 3523 1     SSS$RESIGNAL
2058 3524 1     SSS$CONTINUE
2059 3525 1
2060 3526 1 SIDE EFFECTS:
2061 3527 1
2062 3528 1     May do an UNWIND to let the BASIC-PLUS-2 code process the error.
2063 3529 1     On an UNWIND, will deallocate any heap storage held by its frame.
2064 3530 1
2065 3531 1 --
2066 3532 1
2067 3533 1 BEGIN
2068 3534 1
2069 3535 1 MAP
2070 3536 1     SIGNAL_ARGS : REF BLOCK [0, BYTE],
2071 3537 1     MECHANISM_ARGS : REF BLOCK [0, BYTE];
2072 3538 1
2073 3539 1 BUILTIN
2074 3540 1     CALLG,
2075 3541 1     FP;
2076 3542 1
2077 3543 1 LOCAL
```

! handler for BASIC compiled code
! VAX/VMS signal arguments
! VAX/VMS mechanism arguments

! call with hand-built argument list

```
2078 3544 COND_VAL : BLOCK [4, BYTE],      condition value being signaled
2079 3545 TEMP_COND_VAL : BLOCK [4, BYTE], temp, for copying
2080 3546 NUM_FAD_ARGS,      number of FAD arguments while copying
2081 3547 SCAN_DONE,        used to control copying loop
2082 3548 COPY_LIMIT,      likewise
2083 3549 FMP : REF BLOCK [0, BYTE] FIELD (BSF$FCD), ! frame pointer
2084 3550 LEN_VECTOR : VOLATILE, length of new vector
2085 3551 NEW_VECTOR : REF BLOCK [0, BYTE] VOLATILE, ! the new vector to be signaled
2086 3552 PUTTER,          used to store in new vector
2087 3553 GETTER,          used to fetch from old vector
2088 3554 GET_VM_RESULT,   condition value from calling LIB$GET_VM
2089 3555 USER_RESULT,    condition value from calling BAS$USER_HAND
2090 3556 TOP_LEVEL,     true if we have original jurisdiction
2091 3557 COND_VAL_CHANGE, true if we translated cond
2092 3558 RESTART_IO_FLAG; This is a restartable I/O statement
2093 3559
2094 3560
2095 3561 + The SYSTEM_ERROR cell is 1 if the user specified that system
2096 3562 error handling is to take place. It is set based on a call to
2097 3563 BAS$USER_HAND, and, if set, BAS$USER_HAND will not be called
2098 3564 again for this error.
2099 3565 The GONE_BACK cell is 0 if this is the first instance of the
2100 3566 BASIC handler to see this error, and 1 if we have gone through
2101 3567 a reversion (if only to get traceback).
2102 3568
2103 3569 +
2104 3570 - Declare a handler to clean up heap storage.
2105 3571
2106 3572
2107 3573 ENABLE
2108 3574 HANDLER_HANDLER (LEN_VECTOR, NEW_VECTOR);
2109 3575
2110 3576
2111 3577 + In case we cut back frames, put the signal argument PSL into our frame
2112 3578 so the PSL will be correctly restored. If the emulator is invoked,
2113 3579 it will put an extra frame on the stack and the return PSL will be
2114 3580 incorrect if we fail to do this.
2115 3581
2116 3582
2117 3583 BEGIN
2118 3584 LOCAL
2119 3585 FRAME : REF VECTOR [4,WORD];
2120 3586
2121 3587 FRAME = .FP;
2122 3588 IF .SIGNAL_ARGS [CHF$S_SIG_ARGS] GTR 1
2123 3589 THEN
2124 3590 FRAME [2] = .SIGNAL_ARGS [(SIGNAL_ARGS [CHF$S_SIG_ARGS] * %UPVAL).0,16,0];
2125 3591
2126 3592 END;
2127 3593
2128 3594 +
2129 3595 - Set up a pointer to our frame.
2130 3596 FMP = .MECHANISM_ARGS [CHF$S_MCH_FRAME];
2131 3597
2132 3598
2133 3599 + Remember whether we are the first handler to process this error,
2134 3600 and tell deeper handlers that they are not.
```

```

2135 3601 21- TOP_LEVEL = (IF (.GONE_BACK) THEN 0 ELSE 1);
2136 3602 GONE_BACK = 1;
2137 3603
2138 3604
2139 3605 21- Check for certain non-BASIC errors. Many of these are
2140 3606 converted to their equivalent BASIC error.
2141 3607
2142 3608 COND_VAL = .SIGNAL_ARGS [CHFSL_SIG_NAME];
2143 3609
2144 3610 IF (.COND_VAL [STSSV_FAC_NO] EQL BAS$K_FAC_NO)
2145 3611 THEN
2146 3612 COND_VAL_CHANGE = 0
2147 3613 ELSE
2148 3614 BEGIN
2149 3615
2150 3616 CASE LIB$MATCH COND (COND_VAL,
2151 3617 XREF (MTH$_SQUROONEG),
2152 3618 XREF (MTH$_LOGZERNEG),
2153 3619 XREF (MTH$_FLOOVEMAT),
2154 3620 XREF (SS$_FLTDIV),
2155 3621 XREF (SS$_INTDIV),
2156 3622 XREF (SS$_FLTQVF),
2157 3623 XREF (SS$_INTQVF),
2158 3624 XREF (MTH$_SIGLOSMAT),
2159 3625 XREF (MTH$_UNDEXP),
2160 3626 XREF (SS$_ACCVIO),
2161 3627 XREF (SS$_ROPRAND),
2162 3628 XREF (SS$_SUBRNG),
2163 3629 XREF (STR$_INSVIRMEM),
2164 3630 XREF (STR$_DIVBY_ZER),
2165 3631 XREF (OTSS_IO_CONCLO),
2166 3632 XREF (SS$_ONWIND),
2167 3633 XREF (STR$_STRTOOLON),
2168 3634 XREF (SS$_FLTDIV_F),
2169 3635 XREF (SS$_FLTQVF_F),
2170 3636 XREF (SS$_DECOVFT)
2171 3637 FROM 0 TO 20 OF
2172 3638 SET
2173 3639
2174 3640 [0] :
2175 3641 COND_VAL_CHANGE = 0;
2176 3642 ! none of the above, don't translate
2177 3643
2178 3644 [1] :
2179 3645 BEGIN
2180 3646 COND_VAL = BAS$$COND_VAL (BAS$K_I$MASQUROO);
2181 3647 COND_VAL_CHANGE = 1;
2182 3648 END;
2183 3649
2184 3650 [2] :
2185 3651 BEGIN
2186 3652 COND_VAL = BAS$$COND_VAL (BAS$K_I$LLARGLOG);
2187 3653 COND_VAL_CHANGE = 1;
2188 3654 END;
2189 3655
2190 3656 [4, 5, 14, 18] :
2191 3657 BEGIN
2192 3658 COND_VAL = BAS$$COND_VAL (BAS$K_DIVBY_ZER);

```

```

1 = negative square root
2 = negative or zero log
3 = floating overflow (EXP or TAN)
4 = floating divide by zero
5 = integer divide by zero
6 = floating overflow
7 = integer overflow
8 = significance lost in math library
9 = undefined ** operation
10 = access violation
11 = reserved (floating) operand
12 = subscript out of range
13 = insufficient virtual memory (strings)
14 = String divide by zero
15 = I/O continued to closed file
16 = unwinding through this frame
17 = String too long (greater than 65535)
18 = floating divide by zero fault
19 = floating overflow fault
20 = decimal overflow

```

```
2192 3658 COND_VAL_CHANGE = 1;
2193 3659 END;
2194 3660
2195 3661 [7] :
2196 3662 BEGIN ! integer overflow
2197 3663 COND_VAL = BAS$$COND_VAL (BAS$K_INTERR);
2198 3664 COND_VAL_CHANGE = 1;
2199 3665 END;
2200 3666
2201 3667 [10] :
2202 3668 BEGIN ! invalid address
2203 3669 COND_VAL = BAS$$COND_VAL (BAS$K_MEMMANVIO);
2204 3670 COND_VAL_CHANGE = 1;
2205 3671 END;
2206 3672
2207 3673 [11] :
2208 3674 BEGIN ! reserved operand--fixup to zero
2209 3675
2210 3676 LOCAL
2211 3677 FIXUP_RESULT;
2212 3678
2213 3679 FIXUP_RESULT = LIB$FIXUP_FLT (.SIGNAL_ARGS, .MECHANISM_ARGS);
2214 3680
2215 3681 !+ If the fixup attempt fails, then this handler fails.
2216 3682 !-
2217 3683
2218 3684 IF ( NOT .FIXUP_RESULT)
2219 3685 THEN
2220 3686 BEGIN
2221 3687 GONE_BACK = (IF (.TOP_LEVEL) THEN 0 ELSE 1);
2222 3688 RETURN (.FIXUP_RESULT);
2223 3689 END;
2224 3690
2225 3691 !+ Convert the signal to the catch-all floating point warning for BASIC.
2226 3692 !-
2227 3693
2228 3694 COND_VAL = BAS$$COND_VAL (BAS$K_FLOPOIERR);
2229 3695 COND_VAL_CHANGE = 1;
2230 3696 END;
2231 3697
2232 3698 [3, 6, 8, 9, 19] :
2233 3699 BEGIN ! all other math errors
2234 3700 COND_VAL = BAS$$COND_VAL (BAS$K_FLOPOIERR);
2235 3701 COND_VAL_CHANGE = 1;
2236 3702 END;
2237 3703
2238 3704 [12] :
2239 3705 BEGIN ! subscript out of range
2240 3706 COND_VAL = BAS$$COND_VAL (BAS$K_SUBOUTRAN);
2241 3707 COND_VAL_CHANGE = 1;
2242 3708 END;
2243 3709
2244 3710 [13] :
2245 3711 BEGIN ! String package ran out of memory
2246 3712 COND_VAL = BAS$$COND_VAL (BAS$K_MAXMEMEXC);
2247 3713 COND_VAL_CHANGE = 1;
2248 3714 END;
```



```
2249 3715 3
2250 3716 3
2251 3717 4
2252 3718 4
2253 3719 4
2254 3720 4
2255 3721 4
2256 3722 4
2257 3723 4
2258 3724 4
2259 3725 4
2260 3726 4
2261 3727 4
2262 3728 4
2263 3729 4
2264 3730 4
2265 3731 4
2266 3732 4
2267 3733 4
2268 3734 4
2269 3735 4
2270 3736 4
2271 3737 4
2272 3738 4
2273 3739 4
2274 3740 4
2275 3741 4
2276 3742 4
2277 3743 4
2278 3744 4
2279 3745 4
2280 3746 4
2281 3747 4
2282 3748 4
2283 3749 4
2284 3750 4
2285 3751 4
2286 3752 4
2287 3753 4
2288 3754 4
2289 3755 4
2290 3756 4
2291 3757 4
2292 3758 4
2293 3759 4
2294 3760 4
2295 3761 4
2296 3762 4
2297 3763 4
2298 3764 4
2299 3765 4
2300 3766 4
2301 3767 4
2302 3768 4
2303 3769 4
2304 3770 4
2305 3771 2

[15] :
      BEGIN
      !
      ! I/O continued to closed file. This happens when a function is called in
      ! an I/O list (for example, to evaluate a subscript) and the function
      ! closes the I/O channel that the I/O list is operating on. For compatability
      ! with BASIC-PLUS, give the error message ILLEGAL BYTE COUNT FOR I/O.
      !
      COND_VAL = BAS$$COND_VAL (BAS$K_ILLBYTCOU);
      COND_VAL_CHANGE = 1;
      END;

[16] :
      BEGIN
      ! Unwinding through this frame
      !
      ! We are unwinding through this frame. This may be due to a RESUME
      ! statement cutting back this frame, to the RUN command recovering
      ! from an error, or to a non-BASIC part of the user's program doing
      ! error recovery. Deallocate any heap storage held by this frame.
      !
      BAS$$UNWIND (.FMP);
      END;

[17] :
      BEGIN
      ! String created longer than 65535 characters
      COND_VAL = BAS$$COND_VAL (BAS$K_STRTOOLON);
      COND_VAL_CHANGE = 1;
      END;

[20] :
      BEGIN
      ! Decimal overflow
      COND_VAL = BAS$$COND_VAL (BAS$K_DECERR);
      COND_VAL_CHANGE = 1;
      END;

      TES;

      !
      ! If the translated signal condition is not a BASIC condition,
      ! we can't process it. Return to CHF and indicate that the next
      ! higher frame should be given a chance at it.
      !
      IF (.COND_VAL [ST$V_FAC_NO] NEQ BAS$K_FAC_NO)
      THEN
      BEGIN
      GONE_BACK = (IF (.TOP_LEVEL) THEN 0 ELSE 1);
      RETURN (SS$_RESIGNAL);
      END;
      END;

      !
      ! Give the user's BASIC program a chance to process the error.
      ! If it succeeds, give a success return, otherwise do a special
      ! resignal by extending the signal argument list and signaling again.
```

```

2306 3772 2 :-
2307 3773 2
2308 3774 3 IF (( NOT .SYSTEM_ERROR) AND (.COND_VAL [ST$V_SEVERITY] NEQU ST$K_INFO))
2309 3775 2 THEN
2310 3776 2 USER_RESULT = BAS$USER_HAND (.COND_VAL [ST$V_CODE], .FMP, .MECHANISM_ARGS [CHF$MCH_DEPTH])
2311 3777 2 ELSE
2312 3778 2 USER_RESULT = USER_HAND_FAIL;
2313 3779 2
2314 3780 2 +
2315 3781 2 If the user processes the error to his own satisfaction, skip
2316 3782 2 most of the remainder of this handler.
2317 3783 2 -
2318 3784 2
2319 3785 2 IF (.USER_RESULT NEQ USER_HAND_CONT)
2320 3786 2 THEN
2321 3787 2 BEGIN
2322 3788 2 +
2323 3789 2 If the user specified system handling, set the flag so that the
2324 3790 2 deeper levels of BAS$HANDLER won't call BAS$USER_HAND.
2325 3791 2 -
2326 3792 2
2327 3793 2 IF (.USER_RESULT EQL USER_HAND_FAIL) THEN SYSTEM_ERROR = 1;
2328 3794 2
2329 3795 2 +
2330 3796 2 If we are at the top level purge the terminal's output buffer so
2331 3797 2 that, if a message is printed, it will print after the program's
2332 3798 2 output.
2333 3799 2 -
2334 3800 2
2335 3801 2 IF (.TOP_LEVEL) THEN BAS$PUR_IO_ERR ();
2336 3802 2
2337 3803 2 !<BLF/PAGE>

```

```
2339 3804 +
2340 3805 Append a message about the current frame to the signal argument
2341 3806 list. This requires recopying the list. If we have translated
2342 3807 the signal condition, append the new condition rather than
2343 3808 overwrite the old one, so that a message like 'floating point error'
2344 3809 can have with it a clue as to why it happened.
2345 3810
2346 3811 Compute the length of the new signal argument list.
2347 3812
2348 3813     LEN_VECTOR = (.SIGNAL_ARGS [CHF$S_SIG_ARGS] + 3);
2349 3814
2350 3815     CASE .FMP [BSF$B_PROC_CODE] FROM BSF$K_PROC_MAIN TO BSF$K_PROC_IOL OF
2351 3816     SET
2352 3817         [BSF$K_PROC_MAIN, BSF$K_PROC_SUB, BSF$K_PROC_EXTF] :
2353 3818 +
2354 3819 These frames only have two variables in the FAO list
2355 3820
2356 3821     LEN_VECTOR = .LEN_VECTOR + 2;
2357 3822
2358 3823     [BSF$K_PROC_DEF, BSF$K_PROC_DEFS, BSF$K_PROC_GOSB, BSF$K_PROC_ONER] :
2359 3824 +
2360 3825 These frames have three variables in the FAO list
2361 3826
2362 3827     LEN_VECTOR = .LEN_VECTOR + 3;
2363 3828
2364 3829     [BSF$K_PROC_IOL] :
2365 3830 +
2366 3831 This frame has only one variable in the FAO list
2367 3832
2368 3833     LEN_VECTOR = .LEN_VECTOR + 1;
2369 3834
2370 3835     [OUTRANGE] :
2371 3836 +
2372 3837 If the BSF$B_PROC_CODE byte is out of range then the frame
2373 3838 has been garbaged. There is no point in attempting to continue,
2374 3839 so we merely return to CHF. It is likely that some error message
2375 3840 will be printed.
2376 3841
2377 3842     BEGIN
2378 3843     GONE_BACK = (IF (.TOP_LEVEL) THEN 0 ELSE 1);
2379 3844     RETURN (SS$_RESIGNAL);
2380 3845     END;
2381 3846
2382 3847     TES;
2383 3848
2384 3849 +
2385 3850 Take into account translation of a math error and adding a FAO count
2386 3851 to a short list.
2387 3852
2388 3853     IF (.COND_VAL_CHANGE)
2389 3854     THEN
2390 3855     BEGIN
2391 3856     LEN_VECTOR = .LEN_VECTOR + 6;
2392 3857     END
2393 3858
2394 3859     ELSE
2395 3860
```

```
2396 3861 3 IF (.SIGNAL_ARGS [CHPSL_SIG_ARGS] EQL 3) THEN LEN_VECTOR = .LEN_VECTOR + 1;
2397 3862
2398 3863
2399 3864 * If the argument list is too long, quit. This should only happen if
2400 3865 there is a tall stack of subroutines.
2401 3866
2402 3867
2403 3868 IF (.LEN_VECTOR GTR 250)
2404 3869 THEN
2405 3870 BEGIN
2406 3871 GONE_BACK = (IF (.TOP_LEVEL) THEN 0 ELSE 1);
2407 3872 RETURN (SSS_RESIGNAL);
2408 3873 END;
2409 3874
2410 3875 * Get space to hold the new signal argument list.
2411 3876
2412 3877
2413 3878 IF ( NOT (GET_VM_RESULT = LIB$GET_VM (%REF (.LEN_VECTOR*%UPVAL), NEW_VECTOR)))
2414 3879 THEN
2415 3880
2416 3881 * If we are out of space just quit. This should happen only for
2417 3882 very unreasonable BASIC programs. The BASIC program is given
2418 3883 no chance to recover.
2419 3884
2420 3885
2421 3886 BEGIN
2422 3887 LIB$STOP (.GET_VM_RESULT);
2423 3888 GONE_BACK = (IF (.TOP_LEVEL) THEN 0 ELSE 1);
2424 3889 RETURN (SSS_RESIGNAL);
2425 3890 END;
2426 3891
2427 3892 * Now copy data into the new vector. If we have not translated
2428 3893 the signal condition then our new data goes between the last
2429 3894 of the BASIC data and the first non-BASIC data. If we have
2430 3895 translated the signal condition then our data goes first.
2431 3896
2432 3897
2433 3898 First set the length. Don't count the count longword or the two
2434 3899 trailing longwords.
2435 3900
2436 3901 NEW_VECTOR [0, 0, %BPVAL, 1] = .LEN_VECTOR - 3;
2437 3902 PUTTER = 1;
2438 3903 GETTER = 1;
2439 3904
2440 3905 * If we translated the signal code, store it and a 0 for its FAO count.
2441 3906 Also, store a special message which prints the original PC and PSL.
2442 3907
2443 3908
2444 3909 IF (.COND_VAL_CHANGE)
2445 3910 THEN
2446 3911 BEGIN
2447 3912 NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = .COND_VAL;
2448 3913 PUTTER = .PUTTER + 1;
2449 3914 NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 0;
2450 3915 PUTTER = .PUTTER + 1;
2451 3916 NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$COND_VAL (ERR_TRACE_PCPSL);
2452 3917 ! user PC=!XL, PSC=!XL
```



```
2453 3918 4 PUTTER = .PUTTER + 1;
2454 3919 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 2; ! FAO count
2455 3920 4 PUTTER = .PUTTER + 1;
2456 3921 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = !
2457 3922 4 .SIGNAL_ARGS [(SIGNAL_ARGS [CHFSL_SIG_ARGS] - 1)*%UPVAL, 0, %BPVAL, 0]; ! PC
2458 3923 4 PUTTER = .PUTTER + 1;
2459 3924 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = !
2460 3925 4 .SIGNAL_ARGS [(SIGNAL_ARGS [CHFSL_SIG_ARGS])*%UPVAL, 0, %BPVAL, 0]; ! PSL
2461 3926 4 PUTTER = .PUTTER + 1;
2462 3927 4 END
2463 3928 3 ELSE
2464 3929 4 BEGIN
2465 3930 4 +
2466 3931 4 Otherwise copy all the BASIC data.
2467 3932 4 -
2468 3933 4 SCAN_DONE = 0;
2469 3934 4
2470 3935 4 UNTIL (.SCAN_DONE) DO
2471 3936 5 BEGIN
2472 3937 5 TEMP_COND_VAL = .SIGNAL_ARGS [.GETTER*%UPVAL, 0, %BPVAL, 0];
2473 3938 5
2474 3939 6 IF (.TEMP_COND_VAL [STSSV_FAC_NO] NEQ BAS$K_FAC_NO)
2475 3940 5 THEN
2476 3941 5 SCAN_DONE = 1
2477 3942 5 ELSE
2478 3943 6 BEGIN
2479 3944 6 GETTER = .GETTER + 1;
2480 3945 6 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = .TEMP_COND_VAL;
2481 3946 6 PUTTER = .PUTTER + 1;
2482 3947 6 +
2483 3948 6 Copy the FAO arguments, unless we have reached the end of the list
2484 3949 6 -
2485 3950 6
2486 3951 7 IF (.GETTER NEQ (.SIGNAL_ARGS [CHFSL_SIG_ARGS] - 1))
2487 3952 6 THEN
2488 3953 7 BEGIN
2489 3954 7 NUM_FAO_ARGS = .SIGNAL_ARGS [.GETTER*%UPVAL, 0, %BPVAL, 0];
2490 3955 7 GETTER = .GETTER + 1;
2491 3956 7 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = .NUM_FAO_ARGS;
2492 3957 7 PUTTER = .PUTTER + 1;
2493 3958 7
2494 3959 7 INCR COUNTER FROM 1 TO .NUM_FAO_ARGS DO
2495 3960 8 BEGIN
2496 3961 8 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = !
2497 3962 8 .SIGNAL_ARGS [.GETTER*%UPVAL, 0, %BPVAL, 0];
2498 3963 8 GETTER = .GETTER + 1;
2499 3964 8 PUTTER = .PUTTER + 1;
2500 3965 7 END;
2501 3966 7
2502 3967 7 END
2503 3968 6 ELSE
2504 3969 7 BEGIN
2505 3970 7 +
2506 3971 7 We have reached the end of the list, finding a BASIC condition there.
2507 3972 7 Insert a zero FAO argument count since we will be adding more
2508 3973 7 condition values.
2509 3974 7 -
```

```
2510 3975 7 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 0;
2511 3976 PUTTER = .PUTTER + 1;
2512 3977 END;
2513 3978
2514 3979
2515 3980
2516 3981
2517 3982
2518 3983
2519 3984
2520 3985
2521 3986
2522 3987
2523 3988
2524 3989
2525 3990
2526 3991
2527 3992
2528 3993
2529 3994
2530 3995
2531 3996
2532 3997
2533 3998
2534 3999
2535 4000
2536 4001
2537 4002
2538 4003
2539 4004
2540 4005
2541 4006
2542 4007
2543 4008
2544 4009
2545 4010
2546 4011
2547 4012
2548 4013
2549 4014
2550 4015
2551 4016
2552 4017
2553 4018
2554 4019
2555 4020
2556 4021
2557 4022
2558 4023
2559 4024
2560 4025
2561 4026
2562 4027
2563 4028
2564 4029
2565 4030
2566 4031
```

```
NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 0;
PUTTER = .PUTTER + 1;
END;

+
- Check for the end of the signal arguments.

IF (.GETTER EQLU (.SIGNAL_ARGS [CHFSL_SIG_ARGS] - 1)) THEN SCAN_DONE = 1;
END;
END;
END;

+
- Now put our data in the parameter list we are building.
This data varies depending on the frame type.

CASE .FMP [BSFSB_PROC_CODE] FROM BSFSK_PROC_MAIN TO BSFSK_PROC_IOL OF
SET
[BSFSK_PROC_MAIN] : ! main program
BEGIN
NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$COND_VAL (ERR_TRACE_MAIN);
! message code
PUTTER = .PUTTER + 1;
NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 2; ! number of FAO arguments
PUTTER = .PUTTER + 1;
NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$LINE (.FMP); ! current line number
PUTTER = .PUTTER + 1;
NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$MODULE (.FMP); ! module name
PUTTER = .PUTTER + 1;
END;
[BSFSK_PROC_SUB] : ! external subroutine
BEGIN
NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$COND_VAL (ERR_TRACE_SUB);
! message code
PUTTER = .PUTTER + 1;
NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 2; ! number of FAO arguments
PUTTER = .PUTTER + 1;
NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$LINE (.FMP); ! current line number
PUTTER = .PUTTER + 1;
NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$MODULE (.FMP); ! module name
PUTTER = .PUTTER + 1;
END;
[BSFSK_PROC_EXTF] : ! external function
BEGIN
NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$COND_VAL (ERR_TRACE_EXTF);
! message code
PUTTER = .PUTTER + 1;
NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 2; ! number of FAO arguments
PUTTER = .PUTTER + 1;
```

```
2567 4032 6 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$LINE (.FMP); ! current line number
2568 4033 6 PUTTER = .PUTTER + 1;
2569 4034 6 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$MODULE (.FMP); ! module name
2570 4035 6 PUTTER = .PUTTER + 1;
2571 4036 6 END;
2572 4037 3
2573 4038 3 [BSF$K_PROC_DEF] : ! DEF procedure
2574 4039 4 BEGIN
2575 4040 4 NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$COND_VAL (ERR_TRACE_DEF);
2576 4041 4 ! message code
2577 4042 4 PUTTER = .PUTTER + 1;
2578 4043 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 3; ! number of FAO arguments
2579 4044 4 PUTTER = .PUTTER + 1;
2580 4045 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$LINE (.FMP); ! current line number
2581 4046 4 PUTTER = .PUTTER + 1;
2582 4047 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$FUNCTION (.FMP); ! function name
2583 4048 4 PUTTER = .PUTTER + 1;
2584 4049 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$MODULE (.FMP); ! module name
2585 4050 4 PUTTER = .PUTTER + 1;
2586 4051 3 END;
2587 4052 3
2588 4053 3 [BSF$K_PROC_DEFS] : ! DEF* procedure
2589 4054 4 BEGIN
2590 4055 4 NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$COND_VAL (ERR_TRACE_DEFS);
2591 4056 4 ! message code
2592 4057 4 PUTTER = .PUTTER + 1;
2593 4058 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 3; ! number of FAO arguments
2594 4059 4 PUTTER = .PUTTER + 1;
2595 4060 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$LINE (.FMP); ! current line number
2596 4061 4 PUTTER = .PUTTER + 1;
2597 4062 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$FUNCTION (.FMP); ! function name
2598 4063 4 PUTTER = .PUTTER + 1;
2599 4064 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$MODULE (.FMP); ! module name
2600 4065 4 PUTTER = .PUTTER + 1;
2601 4066 3 END;
2602 4067 3
2603 4068 3 [BSF$K_PROC_GOSB] : ! GOSUB
2604 4069 4 BEGIN
2605 4070 4 NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$COND_VAL (ERR_TRACE_GOSB);
2606 4071 4 ! message code
2607 4072 4 PUTTER = .PUTTER + 1;
2608 4073 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 3; ! number of FAO arguments
2609 4074 4 PUTTER = .PUTTER + 1;
2610 4075 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$LINE (.FMP); ! current line number
2611 4076 4 PUTTER = .PUTTER + 1;
2612 4077 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$FUNCTION (.FMP); ! function number
2613 4078 4 PUTTER = .PUTTER + 1;
2614 4079 4 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$MODULE (.FMP); ! module name
2615 4080 4 PUTTER = .PUTTER + 1;
2616 4081 3 END;
2617 4082 3
2618 4083 3 [BSF$K_PROC_ONER] : ! ON ERROR GOTO
2619 4084 4 BEGIN
2620 4085 4 NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$COND_VAL (ERR_TRACE_ONER);
2621 4086 4 ! message code
2622 4087 4 PUTTER = .PUTTER + 1;
2623 4088 4 NEW_VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 3; ! number of FAO arguments
```



```
2624 4089 6 PUTTER = .PUTTER + 1;
2625 4090 6 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$LINE (.FMP); ! current line number
2626 4091 6 PUTTER = .PUTTER + 1;
2627 4092 6 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$FUNCTION (.FMP); ! function number
2628 4093 6 PUTTER = .PUTTER + 1;
2629 4094 6 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$MODULE (.FMP); ! module name
2630 4095 6 PUTTER = .PUTTER + 1;
2631 4096 6 END;
2632 4097
2633 4098 [BSF$K_PROC_IOL] : ! Immediate mode code
2634 4099 BEGIN
2635 4100 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$COND_VAL (ERR_TRACE_IOLST);
2636 4101 ! message code
2637 4102
2638 4103 PUTTER = .PUTTER + 1;
2639 4104 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = 1; ! number of FA0 arguments
2640 4105 PUTTER = .PUTTER + 1;
2641 4106 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = BAS$$MODULE (.FMP); ! module name
2642 4107 PUTTER = .PUTTER + 1;
2643 4108 END;
2644 4109 [OUTRANGE] :
2645 4110
2646 4111 + If the BSF$B_PROC_CODE byte is out of range then the frame
2647 4112 has been garbaged. There is no point in attempting to continue,
2648 4113 so we merely return to CHF. It is likely that some error message
2649 4114 will be printed.
2650 4115 -
2651 4116 BEGIN
2652 4117 GONE_BACK = (IF (.TOP_LEVEL) THEN 0 ELSE 1);
2653 4118 RETURN (SS$_RESIGNAL);
2654 4119 END;
2655 4120 TES;
2656 4121
2657 4122 +
2658 4123 Copy the remainder of the signal argument list, omitting the
2659 4124 PC and PSL unless we converted the signal condition.
2660 4125 -
2661 4126 COPY_LIMIT = .SIGNAL_ARGS [CHF$S_SIG_ARGS];
2662 4127
2663 4128 IF ( NOT .COND_VAL_CHANGE) THEN COPY_LIMIT = .COPY_LIMIT - 2;
2664 4129
2665 4130 WHILE (.GETTER LEQ .COPY_LIMIT) DO
2666 4131 BEGIN
2667 4132 NEW VECTOR [.PUTTER*%UPVAL, 0, %BPVAL, 0] = .SIGNAL_ARGS [.GETTER*%UPVAL, 0, %BPVAL, 0];
2668 4133 GETTER = .GETTER + 1;
2669 4134 PUTTER = .PUTTER + 1;
2670 4135 END;
2671 4136
2672 4137 +
2673 4138 Set the severity code of the condition which we will signal depending
2674 4139 on the system handling of that signal. However, if this is not an
2675 4140 I/O error from a terminal, don't restart.
2676 4141 -
2677 4142 RESTART_IO_FLAG = 0;
2678 4143
2679 4144 CASE (.ERR_SYSTEM [.COND_VAL [ST$SV_CODE]]) FROM K_SYS_CNT TO K_SYS_RESTART OF
2680 4145 SET
```



```
2681 4146 3
2682 4147 3
2683 4148 4 [K_SYS_CONT] :
2684 4149 4 BEGIN
2685 4150 4 + If the severity is INFO, don't promote it to WARNING.
2686 4151 4 -
2687 4152 4
2688 4153 5 IF (.COND_VAL [STSV_SEVERITY] NEQ STSK_INFO) !
2689 4154 4 THEN
2690 4155 4 COND_VAL [STSV_SEVERITY] = STSK_WARNING;
2691 4156 4
2692 4157 3 END;
2693 4158 3
2694 4159 3 [K_SYS_EXIT] :
2695 4160 4 BEGIN
2696 4161 4 +
2697 4162 4 - If the severity is INFO, don't promote it to SEVERE.
2698 4163 4 -
2699 4164 4
2700 4165 5 IF (.COND_VAL [STSV_SEVERITY] NEQ STSK_INFO) !
2701 4166 4 THEN
2702 4167 4 COND_VAL [STSV_SEVERITY] = STSK_SEVERE;
2703 4168 4
2704 4169 3 END;
2705 4170 3
2706 4171 3 [K_SYS_RESTART] :
2707 4172 4 BEGIN
2708 4173 4
2709 4174 5 IF (LIBSMATCH_COND ((SIGNAL_ARGS [CHFSL_SIG_NAME] + (2*%UPVAL)), %REF (BAS$ON_CHAFIL)))
2710 4175 4 THEN
2711 4176 5 BEGIN
2712 4177 5 +
2713 4178 5 - Because the error code is followed by BAS$ON_CHAFIL the signal must
2714 4179 5 - have been from BAS$$SIGNAL_IO, so this must be an I/O error.
2715 4180 5 - If the I/O is to a terminal, the I/O statement can be restarted.
2716 4181 5 -
2717 4182 5
2718 4183 5 GLOBAL REGISTER
2719 4184 5 CCB = K_CCB_REG : REF BLOCK [,BYTE];
2720 4185 5
2721 4186 5 CCB = .OTSS$A_CUR_LUB;
2722 4187 6 IF (OTSS$TERM_IO () OR
2723 4188 6 .CCB [LUB$V_ANSI])
2724 4189 5 THEN
2725 4190 5 RESTART_IO_FLAG = 1;
2726 4191 5
2727 4192 4 END;
2728 4193 4
2729 4194 5 IF (.RESTART_IO_FLAG)
2730 4195 4 THEN
2731 4196 4 COND_VAL [STSV_SEVERITY] = STSK_WARNING
2732 4197 4 ELSE
2733 4198 4 COND_VAL [STSV_SEVERITY] = STSK_SEVERE;
2734 4199 4
2735 4200 3 END;
2736 4201 3 TES;
2737 4202 3
```

BASERROR
1-074

1 12
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

VAX-11 Bliss-32 V4.0-742
[BASRTL.SRC]BASERROR.B32;1

Page 70
(21)

: 2738
: 2739

4203 3 NEW_VECTOR [1*%UPVAL, 0, %BPVAL, 0] = .COND_VAL;
4204 3 !<BLF/PAGE>

```
2741 4205
2742 4206
2743 4207
2744 4208
2745 4209
2746 4210
2747 4211
2748 4212
2749 4213
2750 4214
2751 4215
2752 4216
2753 4217
2754 4218
2755 4219
2756 4220
2757 4221
2758 4222
2759 4223
2760 4224
2761 4225
2762 4226
2763 4227
2764 4228
2765 4229
2766 4230
2767 4231
2768 4232
2769 4233
2770 4234
2771 4235
2772 4236
2773 4237
2774 4238
2775 4239
2776 4240
2777 4241
2778 4242
2779 4243
2780 4244
2781 4245
2782 4246
2783 4247
2784 4248
2785 4249
2786 4250
2787 4251
2788 4252
2789 4253
2790 4254
2791 4255
2792 4256
2793 4257
2794 4258
2795 4259
2796 4260
2797 4261
```

Now call LIB\$SIGNAL with the argument list we have built. It will invoke this routine recursively for each active frame in the user's BASIC program. Intermediate levels in other languages will be skipped over (provided that the other handlers do not intercept BASIC error codes--if they do, they presumably know what they are doing). The signal argument list includes a traceback in the user's terms.

```
IF ((.ERR_SYSTEM [.COND_VAL [ST$SV_CODE]] EQL K_SYS_EXIT)
AND (.SYSTEM_ERROR)
AND (.COND_VAL [ST$SV_SEVERITY] NEQ ST$K_INFO))
THEN
CALLG (.NEW_VECTOR, LIB$STOP)
ELSE
CALLG (.NEW_VECTOR, LIB$SIGNAL);
```

If we get here the condition is being continued. Either system handling is not being called for, or the system handling is not 'EXIT'.

```
LIB$FREE VM (XREF (.LEN_VECTOR*%UPVAL), NEW_VECTOR);
LEN_VECTOR = 0;
```

If this error is restartable (as determined above) and we are at the top level (that is, the level at which the I/O statement was executed) and no unwind is called for (that is, the user has not executed an error handler, since all error handlers end with a RESUME, which causes an unwind) then restart the I/O statement.

```
IF (.TOP_LEVEL)
THEN
BEGIN
IF ((.RESTART_IO_FLAG) AND (.UNWIND_COUNT EQL 0))
THEN
BEGIN
```

Unwind back to the beginning of the caller's I/O statement. This cannot be done directly, because we don't know where the beginning of the I/O statement is, so we call BAS\$RESTART_IO which puts the I/O system back to the way it was when the I/O list started, and then restarts the I/O list. The call is done through RESTART_IO to get SP restored properly.

```
SYSSUNWIND (MECHANISM_ARGS [CHF$MCH_DEPTH], RESTART_IO);
```

Since we have taken care of this error, clear the error flag.

```
BAS$L_ERRFLG = 0;
END
```

END;

```
2798      END;                                ! of user does not handle the error
2799
2800
2801      + If either this level or a deeper level has requested that the top level
2802      + do an unwind, and if this is the top level, do the unwind.
2803
2804      IF (.TOP_LEVEL AND (.UNWIND_COUNT NEQ 0))
2805      THEN
2806      BEGIN
2807      SY$UNWIND (UNWIND_COUNT, RESTART);
2808      UNWIND_COUNT = 0;
2809      END;
2810
2811      + Set GONE_BACK and SYSTEM_ERROR (own cells) for the previous level
2812      + of BAS$HANDLER.
2813
2814      GONE_BACK = (IF (.TOP_LEVEL) THEN 0 ELSE 1);
2815
2816      IF (.TOP_LEVEL) THEN SYSTEM_ERROR = 0;
2817
2818      RETURN (SS$_CONTINUE);
2819      END;                                ! of BAS$HANDLER
2820
2821
```

				OFFC 00000	.ENTRY	BAS\$HANDLER, Save R2,R3,R4,R5,R6,R7,R8,R9,-;	
			5E	A4 AE 9E 00002	MOVAB	R10,R11	3487
			54	AE 7C 00006	CLRG	-92(SP), SP	
		6D	051D	CF DE 00009	MOVAL	NEW_VECTOR	3533
		51		5D D0 0000E	MOVL	82\$, (FP)	
		55	04	AC D0 00011	MOVL	FP, FRAME	3587
		50		65 D0 00015	MOVL	SIGNAL_ARGS, R5	3588
		01		50 D1 00018	MOVL	(R5), R0	
				05 15 0001B	CMPL	R0, #1	
				05 15 0001B	BLEQ	1\$	
	04	A1		6540 F7 0001D	CVTLW	(R5)[R0], 4(FRAME)	3590
		52	08	AC D0 00022	MOVL	MECHANISM_ARGS, R2	3596
		54	04	A2 D0 00026	MOVL	4(R2), FMP	
		04	00000000	EF E9 0002A	BLBC	GONE_BACK, 2\$	3602
				5A D4 00031	CLRL	TOP_LEVEL	
				03 11 00033	BRB	3\$	
		5A		01 D0 00035	MOVL	#1, TOP_LEVEL	
		EF		01 D0 00038	MOVL	#1, GONE_BACK	3603
	00000000	50		A5 D0 0003F	MOVL	4(R5), COND_VAL	3608
00000000G	8F	52	AE	00 ED 00044	CMPZV	#0, #12, COND_VAL+2, #BAS\$K_FAC_NO	3610
			OC	05 12 0004E	BNEQ	4\$	
				56 D4 00050	CLRL	COND_VAL_CHANGE	3612
				0183 31 00052	BRW	24\$	
	4C	AE	04A4	8F 3C 00055	MOVZWL	#1188, 76(SP)	3636
			4C	AE 9F 0005B	PUSHAB	76(SP)	
	4C	AE	04B4	8F 3C 0005E	MOVZWL	#1204, 76(SP)	3635
			4C	AE 9F 00064	PUSHAB	76(SP)	
	4C	AE	04BC	8F 3C 00067	MOVZWL	#1212, 76(SP)	3634

			4C	AE	9F	0006D	PUSHAB	76(SP)		
			4C	AE	8F	00070	MOVL	#STR\$ STRTOOLON, 76(SP)	3633	
			4C	AE	9F	00078	PUSHAB	76(SP)		
			4C	AE	8F	3C	0007B	MOVZWL	#2336, 76(SP)	3632
			4C	AE	9F	00081	PUSHAB	76(SP)		
			4C	AE	8F	00084	MOVL	#OTS\$ IO_CONCLO, 76(SP)	3631	
			4C	AE	9F	0008C	PUSHAB	76(SP)		
			4C	AE	8F	0008F	MOVL	#STR\$ DIVBY_ZER, 76(SP)	3630	
			4C	AE	9F	00097	PUSHAB	76(SP)		
			4C	AE	8F	0009A	MOVL	#STR\$ INSVIRMEM, 76(SP)	3629	
			4C	AE	9F	000A2	PUSHAB	76(SP)		
			4C	AE	8F	3C	000A5	MOVZWL	#1196, 76(SP)	3628
			4C	AE	9F	000AB	PUSHAB	76(SP)		
			4C	AE	8F	3C	000AE	MOVZWL	#1108, 76(SP)	3627
			4C	AE	9F	000B4	PUSHAB	76(SP)		
			4C	AE	0C	000B7	MOVL	#12, 76(SP)	3626	
			4C	AE	9F	000BB	PUSHAB	76(SP)		
			4C	AE	8F	000BE	MOVL	#MTH\$ UNDEXP, 76(SP)	3625	
			4C	AE	9F	000C6	PUSHAB	76(SP)		
			4C	AE	8F	000C9	MOVL	#MTH\$ SIGLOSMAT, 76(SP)	3624	
			4C	AE	9F	000D1	PUSHAB	76(SP)		
			4C	AE	8F	3C	000D4	MOVZWL	#1148, 76(SP)	3623
			4C	AE	9F	000DA	PUSHAB	76(SP)		
			4C	AE	8F	3C	000DD	MOVZWL	#1164, 76(SP)	3622
			4C	AE	9F	000E3	PUSHAB	76(SP)		
			4C	AE	8F	3C	000E6	MOVZWL	#1156, 76(SP)	3621
			4C	AE	9F	000EC	PUSHAB	76(SP)		
			4C	AE	8F	3C	000EF	MOVZWL	#1172, 76(SP)	3620
			4C	AE	9F	000F5	PUSHAB	76(SP)		
			4C	AE	8F	000F8	MOVL	#MTH\$ FLOOVEMAT, 76(SP)	3619	
			4C	AE	9F	00100	PUSHAB	76(SP)		
			4C	AE	8F	00103	MOVL	#MTH\$ LOGZERNEG, 76(SP)	3618	
			4C	AE	9F	0010B	PUSHAB	76(SP)		
			4C	AE	8F	0010E	MOVL	#MTH\$ SQUROONEG, 76(SP)	3617	
			4C	AE	9F	00116	PUSHAB	76(SP)		
			4C	AE	9F	00119	PUSHAB	COND_VAL	3616	
			F4	AD	9F	0011C	CALLS	#21, LIB\$MATCH_COND		
				15	FB	0011F	CASEL	RO, #0, #20		
				50	CF	00123	.WORD	6\$-5\$,-		
								7\$-5\$,-		
								8\$-5\$,-		
								15\$-5\$,-		
								9\$-5\$,-		
								9\$-5\$,-		
								15\$-5\$,-		
								10\$-5\$,-		
								15\$-5\$,-		
								15\$-5\$,-		
								11\$-5\$,-		
								12\$-5\$,-		
								16\$-5\$,-		
								17\$-5\$,-		
								9\$-5\$,-		
								18\$-5\$,-		
								19\$-5\$,-		
								20\$-5\$,-		
								9\$-5\$,-		
006C	14	00000000G	00							
0040	0034		00							
004C	006C	002E	002A							
007E	0046	003A	003A							
006C	006C	006C	006C							
	003A	0078	0072							
	003A	008F	0084							
			0095							

Address	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418	Op419
---------	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

0018	0012	0018	00000000'	EF	01	D0	00208	28:	MOVL	#1, SYSTEM_ERROR		
001E			00000000G	00	5A	E9	0020F		BLBC	TOP_LEVEL, 29	3801	
				53	00	FB	00212	29:	CALLS	#0, BAS\$SPUR_IO_ERR	3813	
	07	58		AE	65	D0	00219		MOVL	(R5), R3	3815	
	0012			01	A3	9E	0021C		MOVAB	3(R3), LEN_VECTOR		
	0018	0012			A4	8F	00221	30:	CASEB	-27(FMP), #1, #7		
		0018			0012		00226		.WORD	32\$-30\$,-		
					0018		0022E			32\$-30\$,-		
										32\$-30\$,-		
										33\$-30\$,-		
										33\$-30\$,-		
										33\$-30\$,-		
										33\$-30\$,-		
										33\$-30\$,-		
										34\$-30\$,-		
					49	11	00236	31:	BRB	38\$	3844	
		58	AE		02	C0	00238	32:	ADDL2	#2, LEN_VECTOR	3822	
					09	11	0023C		BRB	35\$		
		58	AE		03	C0	0023E	33:	ADDL2	#3, LEN_VECTOR	3828	
					03	11	00242		BRB	35\$		
					58	AE	D6	00244	34:	INCL	LEN_VECTOR	3834
					56	E9	00247	35:	BLBC	COND_VAL_CHANGE, 36\$	3854	
		58	06		06	C0	0024A		ADDL2	#6, LEN_VECTOR	3857	
			AE		08	11	0024E		BRB	37\$	3854	
					53	D1	00250	36:	CMPL	R3, #3	3861	
					03	12	00253		BNEQ	37\$		
			000000FA	8F	58	AE	D6	00255	INCL	LEN_VECTOR		
					58	AE	D1	00258	37:	CMPL	LEN_VECTOR, #250	3868
					1F	14	00260		BGTR	38\$		
					54	AE	9F	00262	PUSHAB	NEW_VECTOR	3879	
50	AE	5C	AE		02	78	00265		ASHL	#2, LEN_VECTOR, 80(SP)		
					50	AE	9F	00268	PUSHAB	80(SP)		
			00000000G	00	02	FB	0026E		CALLS	#2, LIB\$GET_VM		
			OC		50	E8	00275		BLBS	GET_VM_RESULT, 39\$		
					50	DD	00278		PUSHL	GET_VM_RESULT	3887	
			00000000G	00	01	FB	0027A		CALLS	#1, LIB\$STOP		
					00BC	31	00281	38:	BRW	49\$	3888	
54	BE	58	AE		03	C3	00284	39:	SUBL3	#3, LEN_VECTOR, @NEW_VECTOR	3901	
			5B		01	D0	0028A		MOVL	#1, PUTTER	3902	
			52		01	D0	0028D		MOVL	#1, GETTER	3903	
			39		56	E9	00290		BLBC	COND_VAL_CHANGE, 40\$	3909	
		54	BE4B		50	AE	D0	00293	MOVL	COND_VAL, @NEW_VECTOR[PUTTER]	3912	
					5B	D6	00299		INCL	PUTTER	3913	
					54	BE4B	D4	0029B	CLRL	@NEW_VECTOR[PUTTER]	3914	
					5B	D6	0029F		INCL	PUTTER	3915	
			7E		OFF6	8F	3C	002A1	MOVZWL	#4086, -(SP)	3916	
			CF		01	FB	002A6		CALLS	#1, BAS\$COND_VAL		
			54	BE4B	50	D0	002AB		MOVL	R0, @NEW_VECTOR[PUTTER]		
					5B	D6	002B0		INCL	PUTTER	3918	
			54	BE4B	02	D0	002B2		MOVL	#2, @NEW_VECTOR[PUTTER]	3919	
					5B	D6	002B7		INCL	PUTTER	3920	
			54	BE4B	FC	A543	D0	002B9	MOVL	-4(R5)[R3], @NEW_VECTOR[PUTTER]	3922	
					5B	D6	002C0		INCL	PUTTER	3923	
			54	BE4B	6543	D0	002C2		MOVL	(R5)[R3], @NEW_VECTOR[PUTTER]	3925	
					5B	D6	002C8		INCL	PUTTER	3926	
					5F	11	002CA		BRB	47\$	3909	
					57	D4	002CC	40:	CLRL	SCAN_DONE	3933	
			5A		57	E8	002CE	41:	BLBS	SCAN_DONE, 47\$	3935	

00000000G	8F	59	59	6542	D0	002D1	MOVL	(R5)[GETTER], TEMP_COND_VAL	3937	
			0C	10	ED	002D5	CMPZV	#16, #12, TEMP_COND_VAL, #BAS\$K_FAC_NO	3939	
				46	12	002DE	BNEQ	46\$		
		54	BE4B	52	D6	002E0	INCL	GETTER	3944	
				59	D0	002E2	MOVL	TEMP_COND_VAL, @NEW_VECTOR[PUTTER]	3945	
				5B	D6	002E7	INCL	PUTTER	3946	
		50	5B	02	78	002E9	ASHL	#2, PUTTER, R0	3956	
			51	FF	A3	9E	002ED	MOVAB	-1(R3), R1	3951
			51		52	D1	002F1	CMPL	GETTER, R1	
				23	13	002F4	BEOL	44\$		
			58	6542	D0	002F6	MOVL	(R5)[GETTER], NUM_FAO_ARGS	3954	
				52	D6	002FA	INCL	GETTER	3955	
			50	54	AE	C0	002FC	ADDL2	NEW_VECTOR, R0	3956
			60		5B	D0	00300	MOVL	NUM_FAO_ARGS, (R0)	
					5B	D6	00303	INCL	PUTTER	3957
					50	D4	00305	CLRL	COUNTER	3959
				0A	11	00307	BRB	43\$		
		54	BE4B	6542	D0	00309	42\$: MOVL	(R5)[GETTER], @NEW_VECTOR[PUTTER]	3962	
				52	D6	0030F	INCL	GETTER	3963	
				5B	D6	00311	INCL	PUTTER	3964	
		F2	50	58	F3	00313	43\$: AOBLEQ	NUM_FAO_ARGS, COUNTER, 42\$	3959	
				08	11	00317	BRB	45\$	3951	
			50	54	AE	C0	00319	44\$: ADDL2	NEW_VECTOR, R0	3975
				60	D4	0031D	CLRL	(R0)		
				5B	D6	0031F	INCL	PUTTER	3976	
			51	52	D1	00321	45\$: CMPL	GETTER, R1	3983	
				A8	12	00324	BNEQ	41\$		
			57	01	D0	00326	46\$: MOVL	#1, SCAN_DONE		
				A3	11	00329	BRB	41\$	3935	
		07	01	E5	A4	8F	0032B	47\$: CASEB	-27(FMP), #1, #7	3996
0056		0035	002E		0027		00330	48\$: .WORD	52\$-48\$,-	
009F		006B	0064		005D		00338		53\$-48\$,-	
									54\$-48\$,-	
									56\$-48\$,-	
									57\$-48\$,-	
									58\$-48\$,-	
									59\$-48\$,-	
									62\$-48\$	
			04	5A	E9	00340	49\$: BLBC	TOP_LEVEL, 50\$	4117	
				50	D4	00343	CLRL	R0		
				03	11	00345	BRB	51\$		
			50	01	D0	00347	50\$: MOVL	#1, R0		
00000000'			EF	50	D0	0034A	51\$: MOVL	R0, GONE_BACK		
			50	0918	8F	3C	00351	MOVZWL	#2328, R0	4118
					04	00356	RET			
			7E	OFF9	8F	3C	00357	52\$: MOVZWL	#4089, -(SP)	4001
					0C	11	0035C	BRB	55\$	
			7E	OFFA	8F	3C	0035E	53\$: MOVZWL	#4090, -(SP)	4014
					05	11	00363	BRB	55\$	
			7E	OFFB	8F	3C	00365	54\$: MOVZWL	#4091, -(SP)	4027
F82F			CF		01	FB	0036A	55\$: CALLS	#1, BAS\$\$COND_VAL	
			54	BE4B	50	D0	0036F	MOVL	R0, @NEW_VECTOR[PUTTER]	
					5B	D6	00374	INCL	PUTTER	4029
			54	BE4B	02	D0	00376	MOVL	#2, @NEW_VECTOR[PUTTER]	4030
					5B	D6	0037B	INCL	PUTTER	4031
					54	DD	0037D	PUSHL	FMP	4032
F89A			CF		01	FB	0037F	CALLS	#1, BAS\$\$LINE	

				42	11	00384		BRB	61\$		
	7E	OFFC		8F	3C	00386	56\$:	MOVZWL	#4092, -(SP)		4040
				13	11	00388		BRB	60\$		
	7E	OFFD		8F	3C	0038D	57\$:	MOVZWL	#4093, -(SP)		4055
				0C	11	00392		BRB	60\$		
	7E	OFFE		8F	3C	00394	58\$:	MOVZWL	#4094, -(SP)		4070
				05	11	00399		BRB	60\$		
	7E	OFFF		8F	3C	0039B	59\$:	MOVZWL	#4095, -(SP)		4085
F7F9	CF			01	FB	003A0	60\$:	CALLS	#1, BAS\$\$COND_VAL		
54	BE4B			50	D0	003A5		MOVL	R0, @NEW_VECTOR[PUTTER]		
				5B	D6	003AA		INCL	PUTTER		4087
54	BE4B			03	D0	003AC		MOVL	#3, @NEW_VECTOR[PUTTER]		4088
				5B	D6	003B1		INCL	PUTTER		4089
				54	DD	003B3		PUSHL	FMP		4090
F864	CF			01	FB	003B5		CALLS	#1, BAS\$\$LINE		
54	BE4B			50	D0	003BA		MOVL	R0, @NEW_VECTOR[PUTTER]		
				5B	D6	003BF		INCL	PUTTER		4091
				54	DD	003C1		PUSHL	FMP		4092
F866	CF			01	FB	003C3		CALLS	#1, BAS\$\$FUNCTION		
54	BE4B			50	D0	003C8	61\$:	MOVL	R0, @NEW_VECTOR[PUTTER]		
				16	11	003CD		BRB	63\$		4093
	7E	OFF7		8F	3C	003CF	62\$:	MOVZWL	#4087, -(SP)		4100
F7C5	CF			01	FB	003D4		CALLS	#1, BAS\$\$COND_VAL		
54	BE4B			50	D0	003D9		MOVL	R0, @NEW_VECTOR[PUTTER]		
				5B	D6	003DE		INCL	PUTTER		4102
54	BE4B			01	D0	003E0		MOVL	#1, @NEW_VECTOR[PUTTER]		4103
				5B	D6	003E5	63\$:	INCL	PUTTER		4104
				54	DD	003E7		PUSHL	FMP		4105
F86E	CF			01	FB	003E9		CALLS	#1, BAS\$\$MODULE		
54	BE4B			50	D0	003EE		MOVL	R0, @NEW_VECTOR[PUTTER]		
				5B	D6	003F3		INCL	PUTTER		4106
	50			53	D0	003F5		MOVL	R3, COPY_LIMIT		4126
	03			56	E8	003F8		BLBS	COND_VAL_CHANGE, 64\$		4128
	50			02	C2	003FB		SUBL2	#2, COPY_LIMIT		
	50			52	D1	003FE	64\$:	CMPL	GETTER, COPY_LIMIT		4130
				0C	14	00401		BGTR	65\$		
54	BE4B			6542	D0	00403		MOVL	(R5)[GETTER], @NEW_VECTOR[PUTTER]		4132
				52	D6	00409		INCL	GETTER		4133
				5B	D6	0040B		INCL	PUTTER		4134
				EF	11	0040D		BRB	64\$		4130
				53	D4	0040F	65\$:	CLRL	RESTART_IO_FLAG		4142
50	50	AE	0C	03	EF	00411		EXTZV	#3, #12, COND_VAL, R0		4144
		02	01					CASEB	ERR_SYSTEM[R0], #1, #2		
		001A	0010	F65A	CF40	8F 00417	66\$:	.WORD	67\$-66\$,-		
					0006	0041E			68\$-66\$,-		
									69\$-66\$		
									#0, #3, COND_VAL, #3		4153
03	50	AE	03	00	ED	00424	67\$:	CMPZV			
				4C	13	0042A		BEQL	74\$		
				3E	11	0042C		BRB	72\$		4155
03	50	AE	03	00	ED	0042E	68\$:	CMPZV	#0, #3, COND_VAL, #3		4165
				42	13	00434		BEQL	74\$		
				3A	11	00436		BRB	73\$		4167
	4C	AE	00000000G	8F	D0	00438	69\$:	MOVL	#BAS\$ ON_CHAFIL, 76(SP)		4174
				4C	AE	9F 00440		PUSHAB	76(SP)		
				0C	A5	9F 00443		PUSHAB	12(R5)		
	00000000G	00		02	FB	00446		CALLS	#2, LIB\$MATCH_COND		
		19		50	E9	0044D		BLBC	R0, 71\$		

50	AE	03	00	00000000G	00	D0	00450	MOVL	DTSS\$A CUR LUB, CCB	4185	
			00		00	FB	00457	CALLS	#0, DTSS\$TERM_10	4187	
		03	05		50	E8	0045E	BLBS	RO, 70\$		
			AB	A1	04	E1	00461	BBC	#4, -95(CCB), 71\$	4188	
			53		01	D0	00466	MOVL	#1, RESTART_10 FLAG	4190	
			06		53	E9	00469	BLBC	RESTART_10 FLAG, 73\$	4194	
			AE	50	07	8A	0046C	BICB2	#7, COND_VAL	4196	
					06	11	00470	BRB	74\$		
50	AE	03	00		04	F0	00472	INSV	#4, #0, #3, COND_VAL	4198	
			50	54	AE	D0	00478	MOVL	NEW_VECTOR, RO	4203	
			A0	50	AE	D0	0047C	MOVL	COND_VAL, 4(RO)		
50		50	0C		03	EF	00481	EXTZV	#3, #12, COND_VAL, RO	4215	
			02	F5EA	CF	40	91	CMPB	ERR_SYSTEM[RO], #2		
					19	12	0048D	BNEQ	75\$		
			12	00000000'	EF	E9	0048F	BLBC	SYSTEM_ERROR, 75\$	4216	
03		50	03		00	ED	00496	CMPZV	#0, #3, COND_VAL, #3	4217	
					0A	13	0049C	BEQ	75\$		
			00000000G	00	54	BE	FA	0049E	CALLG	@NEW_VECTOR, LIB\$STOP	4219
					08	11	004A6	BRB	76\$		
			00000000G	00	54	BE	FA	004A8	CALLG	@NEW_VECTOR, LIB\$SIGNAL	4221
					54	AE	9F	004B0	PUSHAB	NEW_VECTOR	4228
50	AE		5C	AE	50	AE	9F	004B3	ASHL	#2, LEN_VECTOR, 80(SP)	
					50	AE	9F	004B9	PUSHAB	80(SP)	
			00000000G	00	02	FB	004BC	CALLS	#2, LIB\$FREE_VM		
					58	AE	D4	004C3	CLRL	LEN_VECTOR	4229
			4A		5A	E9	004C6	BLBC	TOP_LEVEL, 79\$	4238	
			1E		53	E9	004C9	BLBC	RESTART_10 FLAG, 77\$	4242	
				00000000'	EF	D5	004CC	TSTL	UNWIND_COUNT		
					16	12	004D2	BNEQ	77\$		
				F7E9	CF	9F	004D4	PUSHAB	RESTART_10	4253	
7E			0B	AC	08	C1	004D8	ADDL3	#8, MECHANISM_ARGS, -(SP)		
			00000000G	00	02	FB	004DD	CALLS	#2, SYSSUNWIND		
					02	FB	004DD	CALLS	#2, SYSSUNWIND		
				00000000'	EF	D4	004E4	CLRL	BAS\$L_ERRFLG	4257	
			26		5A	E9	004EA	BLBC	TOP_LEVEL, 79\$	4269	
				00000000'	EF	D5	004ED	TSTL	UNWIND_COUNT		
					17	13	004F3	BEQ	78\$		
				F7AD	CF	9F	004F5	PUSHAB	RESTART	4272	
				00000000'	EF	9F	004F9	PUSHAB	UNWIND_COUNT		
			00000000G	00	02	FB	004FF	CALLS	#2, SYSSUNWIND		
					02	FB	004FF	CALLS	#2, SYSSUNWIND		
				00000000'	EF	D4	00506	CLRL	UNWIND_COUNT	4273	
			04		5A	E9	0050C	BLBC	TOP_LEVEL, 79\$	4280	
					50	D4	0050F	CLRL	RO		
					03	11	00511	BRB	80\$		
			00000000'	50	01	D0	00513	MOVL	#1, RO		
				EF	50	D0	00516	MOVL	RO, GONE BACK		
			06		5A	E9	0051D	BLBC	TOP_LEVEL, 81\$	4282	
				00000000'	EF	D4	00520	CLRL	SYSTEM_ERROR		
			50		01	D0	00526	MOVL	#1, RO	4284	
					04	00529		RET		4285	
					0000	0052A		.WORD	Save nothing	3533	
			50	08	AC	D0	0052C	MOVL	8(AP), RO		
			50	04	A0	D0	00530	MOVL	4(RO), RO		
				F8	A0	9F	00534	PUSHAB	NEW_VECTOR		
				FC	A0	9F	00537	PUSHAB	LEN_VECTOR		
					D2	DD	0053A	PUSHL	#2		
					5E	DD	0053C	PUSHL	SP		
			7E	04	AC	7D	0053E	MOVQ	4(AP), -(SP)		

BAS\$ERROR
1-074

E 13
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

VAX-11 BLISS-32 V4.0-742
[BASRTL.SRC]BAS\$ERROR.B32;1

Page 79
(22)

F725 CF

03 FB 00542
04 00547

CALLS #3, HANDLER_HANDLER
RET

:

: Routine Size: 1352 bytes, Routine Base: _BAS\$CODE + 068A

: 2822 4286 1

BAS\$ERROR
1-074

F 13
16-Sep-1984 00:23:13 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 11:54:56 [BASRTL.SRC]BAS\$ERROR.B32;1

Page 80
(23)

```
: 2824      4287 1 GLOBAL ROUTINE BAS$ERL =                                ! error line number
: 2825      4288 1
: 2826      4289 1
: 2827      4290 1 ++
: 2828      4291 1 FUNCTIONAL DESCRIPTION:
: 2829      4292 1     Return the line number on which the current error happened.
: 2830      4293 1     If there is no error in progress, return the line number of
: 2831      4294 1     the last error, or 0.
: 2832      4295 1
: 2833      4296 1 FORMAL PARAMETERS:
: 2834      4297 1
: 2835      4298 1     NONE
: 2836      4299 1
: 2837      4300 1 IMPLICIT INPUTS:
: 2838      4301 1
: 2839      4302 1     BAS$L_ERL
: 2840      4303 1
: 2841      4304 1 IMPLICIT OUTPUTS:
: 2842      4305 1
: 2843      4306 1     NONE
: 2844      4307 1
: 2845      4308 1 ROUTINE VALUE:
: 2846      4309 1
: 2847      4310 1     The line number, as a 32-bit binary value.
: 2848      4311 1
: 2849      4312 1 COMPLETION CODES:
: 2850      4313 1
: 2851      4314 1     NONE
: 2852      4315 1
: 2853      4316 1 SIDE EFFECTS:
: 2854      4317 1
: 2855      4318 1     NONE
: 2856      4319 1
: 2857      4320 1 --
: 2858      4321 1
: 2859      4322 2 BEGIN
: 2860      4323 2 RETURN (.BAS$L_ERL);
: 2861      4324 1 END;
```

! of BAS\$ERL

50 00000000' EF 0000 00000
DO 00002
04 00009

.ENTRY BAS\$ERL, Save nothing
MOVL BAS\$L_ERL, R0
RET

: 4287
: 4323
: 4324

; Routine Size: 10 bytes, Routine Base: _BAS\$CODE + 0BD2

; 2862 4325 1

BAS\$ERROR
1-074

G 13
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

VAX-11 Bliss-32 V4.0-742
[BASRTL.SRC]BAS\$ERROR.B32;1

Page 81
(24)

```
2864 4326 1 GLOBAL ROUTINE BAS$ERR = ! error number
2865 4327 1
2866 4328 1
2867 4329 1 **
2868 4330 1 FUNCTIONAL DESCRIPTION:
2869 4331 1 Return the number of the current error.
2870 4332 1 If there is no error in progress, return the number
2871 4333 1 of the last error, or 0.
2872 4334 1
2873 4335 1 FORMAL PARAMETERS:
2874 4336 1 NONE
2875 4337 1
2876 4338 1 IMPLICIT INPUTS:
2877 4339 1
2878 4340 1 BAS$L_ERR
2879 4341 1
2880 4342 1 IMPLICIT OUTPUTS:
2881 4343 1
2882 4344 1 NONE
2883 4345 1
2884 4346 1 ROUTINE VALUE:
2885 4347 1
2886 4348 1 The error number, as a 32-bit binary value.
2887 4349 1
2888 4350 1 COMPLETION CODES:
2889 4351 1
2890 4352 1 NONE
2891 4353 1
2892 4354 1 SIDE EFFECTS:
2893 4355 1
2894 4356 1 NONE
2895 4357 1
2896 4358 1
2897 4359 1 --
2898 4360 1
2899 4361 2 BEGIN
2900 4362 2 RETURN (.BAS$L_ERR);
2901 4363 1 END; ! of BAS$ERR
```

50 00000000' EF 0000 0000
D0 00002
04 00009

ENTRY BAS\$ERR, Save nothing
MOVL BAS\$L_ERR, R0
RET

: 4326
: 4362
: 4363

; Routine Size: 10 bytes, Routine Base: _BAS\$CODE + 0BDC

; 2902 4364 1

```
2904 4365 1 GLOBAL ROUTINE BASERN (
2905 4366 1     DESCRIPTOR
2906 4367 1     ) =
2907 4368 1
2908 4369 1
2909 4370 1
2910 4371 1
2911 4372 1
2912 4373 1
2913 4374 1
2914 4375 1
2915 4376 1
2916 4377 1
2917 4378 1
2918 4379 1
2919 4380 1
2920 4381 1
2921 4382 1
2922 4383 1
2923 4384 1
2924 4385 1
2925 4386 1
2926 4387 1
2927 4388 1
2928 4389 1
2929 4390 1
2930 4391 1
2931 4392 1
2932 4393 1
2933 4394 1
2934 4395 1
2935 4396 1
2936 4397 1
2937 4398 1
2938 4399 2
2939 4400 2
2940 4401 2
2941 4402 2
2942 4403 2
2943 4404 2
2944 4405 2
2945 4406 1

    **
    FUNCTIONAL DESCRIPTION:
        Return the name of the module in which the current error
        happened.  If there is no error in progress, return
        the module name for the last error, or a zero-length string.

    FORMAL PARAMETERS:
        DESCRIPTOR.wt.dx  A descriptor into which to write the name of
                           the module.

    IMPLICIT INPUTS:
        BAS$T_ERN

    IMPLICIT OUTPUTS:
        NONE

    COMPLETION CODES:
        Same as for STR$COPY

    SIDE EFFECTS:
        Calls STR$COPY; if it fails, this routine never returns.

    --
    BEGIN
    LOCAL
        COPY_STATUS;

    COPY_STATUS = STR$COPY_DX (.DESCRIPTOR, BAS$T_ERN);
    RETURN (COPY_STATUS);
    END;
```

```
! error module name
! where to write the name
```

```
! of BASERN
```

```

                                0000 00000
                                04 C2 00002
                                EF 9F 00005
                                AC DD 0000B
00000000G 00 02 FB 0000E
                                50 D0 00015
                                6E 9E 00018
                                04 0001B

.ENTRY BASERN, Save nothing
SUBL2 #4, SP
PUSHAB BAS$T_ERN
PUSHL DESCRIPTOR
CALLS #2, STR$COPY_DX
MOVL R0, COPY_STATUS
MOVAB COPY_STATUS, R0
RET
```

```
; Routine Size: 28 bytes, Routine Base: _BAS$CODE + 0BE6
```

```
4365
4404
4405
4406
```

BASERROR
1-074

1 13
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

VAX-11 Bliss-32 V4.0-742
[BASRTL.SRC]BASERROR.B32;1

Page 83
(25)

: 2946

4407 1

```

2948 4408 1 GLOBAL ROUTINE BASSERT (
2949 4409 1     DESCRIPTOR,
2950 4410 1     ERRNO
2951 4411 1 ) =
2952 4412 1
2953 4413 1 ++
2954 4414 1 FUNCTIONAL DESCRIPTION:
2955 4415 1
2956 4416 1     Return the text of an error message. If the error number is
2957 4417 1     unreasonable, the result is undefined. The first character of
2958 4418 1     the message indicates the severity: '?' is a fatal error,
2959 4419 1     'x' is a warning, and all other messages start with a space.
2960 4420 1
2961 4421 1 FORMAL PARAMETERS:
2962 4422 1
2963 4423 1     DESCRIPTOR.wt.d    A descriptor into which to write the text of
2964 4424 1                       the message.
2965 4425 1     ERRNO.rl.v         The error number for which we want the text.
2966 4426 1
2967 4427 1 IMPLICIT INPUTS:
2968 4428 1
2969 4429 1     The system error message file, SYS$MESSAGE:SYSMSG.EXE
2970 4430 1
2971 4431 1 IMPLICIT OUTPUTS:
2972 4432 1
2973 4433 1     NONE
2974 4434 1
2975 4435 1 COMPLETION CODES:
2976 4436 1
2977 4437 1     Same as STR$CONCAT
2978 4438 1
2979 4439 1 SIDE EFFECTS:
2980 4440 1
2981 4441 1     Calls several system functions. If any fail, this routine
2982 4442 1     never returns.
2983 4443 1
2984 4444 1 --
2985 4445 1
2986 4446 2 BEGIN
2987 4447 2
2988 4448 2 LOCAL
2989 4449 2     CONCAT_RESULT,
2990 4450 2     GETMSG_RESULT,
2991 4451 2     LOCAL_DESCRIPTOR : BLOCK [8, BYTE],
2992 4452 2     Q_DESC : BLOCK [8, BYTE],
2993 4453 2     Q_BUF : VECTOR [1, BYTE],
2994 4454 2     LOCAL_BUF : VECTOR [256, BYTE],
2995 4455 2     DUMMY;
2996 4456 2
2997 4457 2 ++
2998 4458 2 Set up the local descriptor.
2999 4459 2
3000 4460 2 LOCAL_DESCRIPTOR [DSC$W_LENGTH] = 256;
3001 4461 2 LOCAL_DESCRIPTOR [DSC$B_DTYPE] = DSC$K_DTYPE_T;
3002 4462 2 LOCAL_DESCRIPTOR [DSC$B_CLASS] = DSC$K_CLASS_S;
3003 4463 2 LOCAL_DESCRIPTOR [DSC$A_POINTER] = LOCAL_BUF;
3004 4464 2 ++

```

! error text
! where to put text
! error number

! Status from STR\$CONCAT
! remembers status of SYS\$GETMSG
! message descriptor
! Points to ?, x or space
! Holds the ?, x or space
! Buffer got SYS\$GETMSG
! used to discard last value from SYS\$GETMSG


```

3005 4465 2 Get the message text from SYSS$MESSAGE:SYSMSG.EXE
3006 4466
3007 4467 GETMSG_RESULT = SYSS$GETMSG (BAS$$COND_VAL (.ERRNO), LOCAL_DESCRIP, LOCAL_DESCRIP, 1, DUMMY);
3008 4468
3009 4469 IF ( NOT .GETMSG_RESULT) THEN LIB$STOP (.GETMSG_RESULT);
3010 4470
3011 4471
3012 4472 Copy the message text to the user's string, concatenating a ?, ! or
3013 4473 space onto its front to indicate the severity of the error.
3014 4474
3015 4475 Q_DESC [DSC$W_LENGTH] = 1;
3016 4476 Q_DESC [DSC$B_DTYPE] = DSC$K_DTYPE_T;
3017 4477 Q_DESC [DSC$B_CLASS] = DSC$K_CLASS_S;
3018 4478 Q_DESC [DSC$A_POINTER] = Q_BUF [0];
3019 4479 Q_BUF [0] =
3020 4480 BEGIN
3021 4481
3022 4482 CASE .ERR_SEVERITY [.ERRNO] FROM STS$K_WARNING TO STS$K_SEVERE OF
3023 4483 SET
3024 4484 [STS$K_WARNING] :
3025 4485 %C'!';
3026 4486
3027 4487 [STS$K_SEVERE, STS$K_ERROR] :
3028 4488 %C'?';
3029 4489
3030 4490 [INRANGE, OUTRANGE] :
3031 4491 %C'!';
3032 4492
3033 4493 TES
3034 4494
3035 4495 END;
3036 4496 CONCAT_RESULT = STR$CONCAT (.DESCRIP, Q_DESC, LOCAL_DESCRIP);
3037 4497 RETURN (.CONCAT_RESULT);
3038 4498 END;

```

! of BAS\$ERT

			0000 00000	.ENTRY	BAS\$ERT, Save nothing	4408
	5E	FEEB	CE 9E 00002	MOVAB	-280(SP), SP	
F8	AD	010E0100	8F D0 00007	MOVL	#17694976, LOCAL_DESCRIP	4460
FC	AD	08	AE 9E 0000F	MOVAB	LOCAL_BUF, LOCAL_DESCRIP+4	4463
			5E DD 00014	PUSHL	SP	4467
			01 DD 00016	PUSHL	#1	
		F8	AD 9F 00018	PUSHAB	LOCAL_DESCRIP	
		F8	AD 9F 0001B	PUSHAB	LOCAL_DESCRIP	
		08	AC DD 0001E	PUSHL	ERRNO	
F600	CF		01 FB 00021	CALLS	#1, BAS\$\$COND_VAL	
			50 DD 00026	PUSHL	RO	
00000000G	00		05 FB 00028	CALLS	#5, SYSS\$GETMSG	
	09		50 E8 0002F	BLBS	GETMSG_RESULT, 18	4469
			50 DD 00032	PUSHL	GETMSG_RESULT	
00000000G	00		01 FB 00034	CALLS	#1, LIB\$STOP	
F0	AD	010E0001	8F D0 0003B 18:	MOVL	#17694721, Q_DESC	4475
F4	AD	04	AE 9E 00043	MOVAB	Q_BUF, Q_DESC+4	4478
	50	F3B2	CF 9E 00048	MOVAB	ERR_SEVERITY, RO	4482

BASERROR
1-074

L 13
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

VAX-11 Bliss-32 V4.0-742
[BASRTL.SRC]BASERROR.B32;1

Page 86
(26)

000A	04 0014	00 000A	08 BC40 BF 00040 000F 00053 2%: 0014 00058	CASEB .WORD	@ERRNO[R0], #0, #4 4%-2%,- 3%-2%,- 5%-2%,- 3%-2%,- 5%-2%,- 5%-2%
		50	20 D0 00050 3%: 08 11 00060	MOVL	#32, R0
		50	25 D0 00062 4%: 03 11 00065	BRB	6%
		50	3F D0 00067 5%: 50 90 0006A 6%:	MOVL	#37, R0
04	AE		F8 AD 9F 0006E F0 AD 9F 00071	BRB	6%
			04 AC DD 00074	MOVL	#63, R0
		00000000G 00	03 FB 00077	MOVB	R0, Q_BUF
			04 0007E	PUSHAB	LOCAL-DESCRIP
				PUSHAB	Q_DESC
				PUSHL	DESCRIP
				CALLS	#3, STR\$CONCAT
				RET	

4480
4496
4498

; Routine Size: 127 bytes, Routine Base: _BAS\$CODE + 0C02

; 3039 4499 1

```

3041 4500 1 GLOBAL ROUTINE BAS$ERROR (          ! signal an error
3042 4501 1     ERRNO                          ! error number
3043 4502 1     ) : NOVALUE =
3044 4503 1
3045 4504 1
3046 4505 1 **
3047 4506 1 FUNCTIONAL DESCRIPTION:
3048 4507 1     Signals an error. This is called from the compiled code when
3049 4508 1     an error condition is detected in line. It can also be called
3050 4509 1     by user code directly.
3051 4510 1
3052 4511 1 FORMAL PARAMETERS:
3053 4512 1
3054 4513 1     ERRNO.rl.v      The error number which we want to signal.
3055 4514 1
3056 4515 1 IMPLICIT INPUTS:
3057 4516 1
3058 4517 1     NONE
3059 4518 1
3060 4519 1 IMPLICIT OUTPUTS:
3061 4520 1
3062 4521 1     NONE
3063 4522 1
3064 4523 1 ROUTINE VALUE:
3065 4524 1
3066 4525 1     NONE
3067 4526 1
3068 4527 1 COMPLETION CODES:
3069 4528 1
3070 4529 1     NONE
3071 4530 1
3072 4531 1 SIDE EFFECTS:
3073 4532 1
3074 4533 1     May never return to its caller.
3075 4534 1
3076 4535 1 --
3077 4536 1
3078 4537 2 BEGIN
3079 4538 2
3080 4539 2 + Don't allow error numbers above 255, since we keep some internal
3081 4540 2 messages up there.
3082 4541 2 -
3083 4542 2
3084 4543 2 IF (.ERRNO LEQU 255) THEN BAS$$SIGNAL (.ERRNO) ELSE BAS$$STOP (BAS$K_PROLOSSOR);
3085 4544 2
3086 4545 1 END;                                ! of BAS$ERROR

```

000000FF	8F	04	AC	D1	00002	.ENTRY	BAS\$ERROR, Save nothing
			09	1A	0000A	CMPL	ERRNO, #255
		04	AC	DD	0000C	BGTRU	1\$
F56B	CF		01	FB	0000F	PUSHL	ERRNO
				04	00014	CALLS	#1, BAS\$\$SIGNAL
						RET	

4500
4543
.....

BAS\$ERROR
1-074

N 13
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

VAX-11 Bliss-32 V4.0-742
[BASRTL.SRC]BAS\$ERROR.B32;1

Page 88
(27)

F575 7E 00G 8F 9A 00015 1\$: MOVZBL #BAS\$K PROLOSSOR, -(SP)
CF 01 FB 00019 CALLS #1, BAS\$\$\$STOP
04 0001E RET

; Routine Size: 31 bytes. Routine Base: _BAS\$CODE + 0C81

; 3087 4546 1

:
:
: 4545


```
3089 4547 1 GLOBAL ROUTINE BAS$$ERR_INIT : NOVALUE =      ! Initialize error flag
3090 4548 1
3091 4549 1
3092 4550 1 **
3093 4551 1 FUNCTIONAL DESCRIPTION:
3094 4552 1       Initialize the error status. This is used by the RUN command in case the user
3095 4553 1       does a RUN following an error.
3096 4554 1
3097 4555 1 FORMAL PARAMETERS:
3098 4556 1
3099 4557 1     NONE
3100 4558 1
3101 4559 1 IMPLICIT INPUTS:
3102 4560 1
3103 4561 1     NONE
3104 4562 1
3105 4563 1 IMPLICIT OUTPUTS:
3106 4564 1
3107 4565 1     All of the error cells that the user can see.
3108 4566 1
3109 4567 1 ROUTINE VALUE:
3110 4568 1
3111 4569 1     NONE
3112 4570 1
3113 4571 1 COMPLETION CODES:
3114 4572 1
3115 4573 1     NONE
3116 4574 1
3117 4575 1 SIDE EFFECTS:
3118 4576 1
3119 4577 1     After this routine has been called, we are not in an error handler.
3120 4578 1     Also, the error stack is empty.
3121 4579 1
3122 4580 1 --
3123 4581 1
3124 4582 2 BEGIN
3125 4583 2
3126 4584 2 First, empty the error stack.
3127 4585 2
3128 4586 2
3129 4587 2 WHILE (.ERROR_STACK [0] NEQA .ERROR_STACK [1]) DO
3130 4588 2     BAS$POP_ERR ();
3131 4589 2
3132 4590 2
3133 4591 2 Then make sure we are not in an error routine.
3134 4592 2
3135 4593 2     BAS$L_ERRFLG = 0;
3136 4594 2
3137 4595 2 See to it that all user-visible cells are in their initial states.
3138 4596 2
3139 4597 2     BAS$A_CH_CUR_LN = 0;
3140 4598 2     BAS$L_GOING_BACK = 0;
3141 4599 2     SYSTEM_ERROR = 0;
3142 4600 2     GONE_BACK = 0;
3143 4601 2     BAS$C_ERL = 0;
3144 4602 2     BAS$L_ERR = 0;
3145 4603 2     BAS$T_ERN [DSC$W_LENGTH] = 0;
```

BAS\$ERROR
1-074

C 14
16-Sep-1984 00:23:13 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 11:54:56 [BASRTL.SRC]BAS\$ERROR.B32;1

Page 90
(28)

; 3146 4604 1 END;

! of BAS\$ERR_INIT

			0004	00000		.ENTRY	BAS\$ERR_INIT, Save R2	.. 4547	
	S2	00000000*	EF	9E	00002	MOVAB	ERROR_STACK, R2	..	
04	A2		62	D1	00009	1\$:	CMPL	ERROR_STACK, ERROR_STACK+4	.. 4587
			07	13	0000D		BEQL	2\$..
0000V	CF		00	FB	0000F		CALLS	#0, BAS\$POP_ERR	.. 4588
			F3	11	00014		BRB	1\$..
		0C	A2	7C	00016	2\$:	CLRQ	BAS\$A_CH_CUR_LN	.. 4597
		F8	A2	7C	00019		CLRQ	SYSTEM_ERROR	.. 4599
		E0	A2	7C	0001C		CLRQ	BAS\$L_ERR	.. 4601
		DC	A2	D4	0001F		CLRL	BAS\$L_ERR	.. 4602
		D4	A2	B4	00022		CLRW	BAS\$T_ERN	.. 4603
			04	00025		RET		.. 4604	

; Routine Size: 38 bytes, Routine Base: _BAS\$CODE + 0CA0

; 3147 4605 1

```
.. 3149 4606 1 GLOBAL ROUTINE BASS$PUSH_ERR = ! Push error status
.. 3150 4607 1
.. 3151 4608 1
.. 3152 4609 1 ++
.. 3153 4610 1 FUNCTIONAL DESCRIPTION:
.. 3154 4611 1 Save the error state on the error stack, so a BASIC program can
.. 3155 4612 1 be run independent of the error flag.
.. 3156 4613 1
.. 3157 4614 1 CALLING SEQUENCE:
.. 3158 4615 1
.. 3159 4616 1 CALL BASS$PUSH_ERR ()
.. 3160 4617 1
.. 3161 4618 1 FORMAL PARAMETERS:
.. 3162 4619 1
.. 3163 4620 1 NONE
.. 3164 4621 1
.. 3165 4622 1 IMPLICIT INPUTS:
.. 3166 4623 1
.. 3167 4624 1 The OWN cells which represent the error status.
.. 3168 4625 1
.. 3169 4626 1 IMPLICIT OUTPUTS:
.. 3170 4627 1
.. 3171 4628 1 The error stack
.. 3172 4629 1
.. 3173 4630 1 SIDE EFFECTS:
.. 3174 4631 1
.. 3175 4632 1 Calls LIB$GET_VM to get virtual memory.
.. 3176 4633 1 --
.. 3177 4634 1
.. 3178 4635 2 BEGIN
.. 3179 4636 2
.. 3180 4637 2 BUILTIN
.. 3181 4638 2 INSQUE;
.. 3182 4639 2
.. 3183 4640 2 LOCAL
.. 3184 4641 2
.. 3185 4642 2 Declare the pointer to the block to push.
.. 3186 4643 2
.. 3187 4644 2 PUSH : REF BLOCK [PUSH$K_LENGTH, BYTE] FIELD (PUSH_ITEM);
.. 3188 4645 2
.. 3189 4646 2
.. 3190 4647 2 If this is the first PUSH, initialize the queue.
.. 3191 4648 2
.. 3192 4649 2
.. 3193 4650 2 IF ( NOT .ERROR_STACK_INI)
.. 3194 4651 2 THEN
.. 3195 4652 2 BEGIN
.. 3196 4653 2
.. 3197 4654 2 LOCAL
.. 3198 4655 2 AST_STATUS;
.. 3199 4656 2
.. 3200 4657 2 AST_STATUS = $SETAST (ENBFLG = 0);
.. 3201 4658 2
.. 3202 4659 2 IF ( NOT .ERROR_STACK_INI)
.. 3203 4660 2 THEN
.. 3204 4661 2 BEGIN
.. 3205 4662 2 ERROR_STACK [0] = ERROR_STACK [1] = ERROR_STACK [0];
```

```
3206 4663 ERROR_STACK_INIT = 1;
3207 4664 END;
3208 4665
3209 4666 IF (.AST_STATUS EQL SSS_WASSET) THEN $SETAST (ENBFLG = 1);
3210 4667
3211 4668 END;
3212 4669
3213 4670 !+
3214 4671 !- Get virtual memory to hold the error state.
3215 4672 !-
3216 4673 BEGIN
3217 4674
3218 4675 LOCAL
3219 4676 GET_VM_RESULT;
3220 4677
3221 4678 GET_VM_RESULT = LIB$GET_VM (%REF (PUSH$K_LENGTH), PUSH);
3222 4679
3223 4680 IF ( NOT .GET_VM_RESULT) THEN BAS$$STOP (BAS$K_MAXMEMEXC);
3224 4681
3225 4682 END;
3226 4683 !+
3227 4684 !- Fill in.
3228 4685 !-
3229 4686 PUSH [PUSH$L_ERRFLG] = .BAS$L_ERRFLG;
3230 4687 CH$MOVE (8, BAS$T_ERN, PUSH [PUSH$T_ERN]);
3231 4688 PUSH [PUSH$L_ERR] = .BAS$L_ERR;
3232 4689 PUSH [PUSH$L_ERL] = .BAS$L_ERL;
3233 4690 PUSH [PUSH$L_HGH_LVL] = .HIGHEST_LEVEL;
3234 4691 PUSH [PUSH$A_HGH_FMP] = .HIGHEST_FMP;
3235 4692 PUSH [PUSH$L_ACC_LVL] = .ACCUM_LEVEL;
3236 4693 PUSH [PUSH$L_UNW_CNT] = .UNWIND_COUNT;
3237 4694 PUSH [PUSH$L_SYS_ERR] = .SYSTEM_ERROR;
3238 4695 PUSH [PUSH$L_GONE_BAK] = .GONE_BACK;
3239 4696 PUSH [PUSH$A_CUR [IN]] = .BAS$A_CH_CUR LN;
3240 4697 PUSH [PUSH$L_GOING_BACK] = .BAS$L_GOING_BACK;
3241 4698 PUSH [PUSH$A_RESTART] = .BAS$A_RESTART;
3242 4699 !+
3243 4700 !- Put this item on the error stack.
3244 4701 !-
3245 4702 INSQUE (.PUSH, ERROR_STACK);
3246 4703 !+
3247 4704 !- Make sure there is no error outstanding.
3248 4705 !-
3249 4706 BAS$L_ERRFLG = 0;
3250 4707 BAS$A_CH_CUR LN = 0;
3251 4708 BAS$L_GOING_BACK = 0;
3252 4709 SYSTEM_ERROR = 0;
3253 4710 GONE_BACK = 0;
3254 4711 !+
3255 4712 !- Successful completion.
3256 4713 !-
3257 4714 RETURN (SS$NORMAL);
3258 4715 END;
```

! of routine BAS\$PUSH_ERR

.EXTRN SYS\$SETAST

				01FC 00000	.ENTRY	BASSPUSH ERR, Save R2,R3,R4,R5,R6,R7,R8	4606
	58	00000000G	00	9E 00002	MOVAB	SYSSSETAST, R8	
	57	000000000	EF	9E 00009	MOVAB	ERROR_STACK_INI, R7	
	5E		08	C2 00010	SUBL2	#8, SP	
	21		67	E8 00013	BLBS	ERROR_STACK_INI, 2\$	4650
			7E	D4 00016	CLRL	-(SP)	4657
	68		01	FB 00018	CALLS	#1, SYSSSETAST	
	0F		67	E8 0001B	BLBS	ERROR_STACK_INI, 1\$	4659
	51	F8	A7	9E 0001E	MOVAB	ERROR_STACK, R1	4662
FC	A7		51	DO 00022	MOVL	R1, ERROR_STACK+4	
F8	A7		51	DO 00026	MOVL	R1, ERROR_STACK	
	67		01	DO 0002A	MOVL	#1, ERROR_STACK_INI	4663
	09		50	D1 0002D	CMPL	AST_STATUS, #9	4666
			05	12 00030	BNEQ	2\$	
			01	DD 00032	PUSHL	#1	
	68		01	FB 00034	CALLS	#1, SYSSSETAST	
		04	AE	9F 00037	PUSHAB	PUSH	4678
04	AE	40	8F	9A 0003A	MOVZBL	#64, 4(SP)	
		04	AE	9F 0003F	PUSHAB	4(SP)	
00000000G	00		02	FB 00042	CALLS	#2, LIB\$GET VM	
	09		50	E8 00049	BLBS	GET VM_RESULT, 3\$	4680
	7E	00G	8F	9A 0004C	MOVZBL	#BASSK_MAXMEMEXC, -(SP)	
F4F9	CF		01	FB 00050	CALLS	#1, BASS\$STOP	
	56	04	AE	DO 00055	MOVL	PUSH, R6	4686
OC	A6	DC	A7	DO 00059	MOVL	BASSL_ERRFLG, 8(R6)	
	CC		08	28 0005E	MOVC3	#8, BASST_ERN, 12(R6)	4687
	14	A6	A7	7D 00064	MOVQ	BASSL_ERR, 20(R6)	4688
	1C	A6	E0	A7 7D 00069	MOVQ	HIGHEST_LEVEL, 28(R6)	4690
	24	A6	E8	A7 7D 0006E	MOVQ	ACCUM_LEVEL, 36(R6)	4692
	2C	A6	F0	A7 7D 00073	MOVQ	SYSTEM_ERROR, 44(R6)	4694
	34	A6	04	A7 7D 00078	MOVQ	BASSA_CH_CUR_LN, 52(R6)	4696
	3C	A6	0C	A7 DO 0007D	MOVL	BASSA_RESTART, 60(R6)	4698
	F8	A7	66	0E 00082	INSQUE	(R6), ERROR_STACK	4702
		DC	A7	D4 00086	CLRL	BASSL_ERRFLG	4706
		04	A7	7C 00089	CLRQ	BASSA_CH_CUR_LN	4707
		F0	A7	7C 0008C	CLRQ	SYSTEM_ERROR	4709
	50		01	DO 0008F	MOVL	#1, R0	4714
			04	00092	RET		4715

: Routine Size: 147 bytes, Routine Base: _BAS\$CODE + 0CC6

: 3259 4716 1

```

3261 4717 1 GLOBAL ROUTINE BASSPOP_ERR =                ! Pop error status
3262 4718 1
3263 4719 1 ++
3264 4720 1 FUNCTIONAL DESCRIPTION:
3265 4721 1
3266 4722 1         Restore the error state from the error stack.
3267 4723 1
3268 4724 1 CALLING SEQUENCE:
3269 4725 1
3270 4726 1         CALL BASSPOP_ERR ()
3271 4727 1
3272 4728 1 FORMAL PARAMETERS:
3273 4729 1
3274 4730 1         NONE
3275 4731 1
3276 4732 1 IMPLICIT INPUTS:
3277 4733 1
3278 4734 1         The error stack.
3279 4735 1
3280 4736 1 IMPLICIT OUTPUTS:
3281 4737 1
3282 4738 1         The OWN storage which represents the error state.
3283 4739 1
3284 4740 1 SIDE EFFECTS:
3285 4741 1
3286 4742 1         Calls LIB$FREE_VM to free virtual memory.
3287 4743 1 --
3288 4744 1
3289 4745 2 BEGIN
3290 4746 2
3291 4747 2 BUILTIN
3292 4748 2 REMQUE;
3293 4749 2
3294 4750 2 LOCAL
3295 4751 2     PUSH : REF BLOCK [PUSH$K_LENGTH, BYTE] FIELD (PUSH_ITEM);
3296 4752 2
3297 4753 2 ++
3298 4754 2 Get an item off the error stack. It had better be there.
3299 4755 2 --
3300 4756 2
3301 4757 2     IF (REMQUE (.ERROR_STACK [0], PUSH)) THEN BASS$STOP (BASS$K_PROLOSSOR);
3302 4758 2
3303 4759 2 ++
3304 4760 2 Copy the data from the stack into the OWN cells, thus reestablishing
3305 4761 2 the error environment at the time of the PUSH.
3306 4762 2 --
3307 4763 2     BASS$ERRFLG = .PUSH [PUSH$K_ERRFLG];
3308 4764 2     CH$MOVE (8, PUSH [PUSH$K_ERR], BASS$ERN);
3309 4765 2     BASS$ERR = .PUSH [PUSH$K_ERR];
3310 4766 2     BASS$ERL = .PUSH [PUSH$K_ERL];
3311 4767 2     HIGHEST_LEVEL = .PUSH [PUSH$K_HGH_LVL];
3312 4768 2     HIGHEST_FMP = .PUSH [PUSH$K_HGH_FMP];
3313 4769 2     ACCUM_LEVEL = .PUSH [PUSH$K_ACC_LVL];
3314 4770 2     UNWIND_COUNT = .PUSH [PUSH$K_UNW_CNT];
3315 4771 2     SYSTEM_ERROR = .PUSH [PUSH$K_SYS_ERR];
3316 4772 2     GONE_BACK = .PUSH [PUSH$K_GONE_BAK];
3317 4773 2     BASS$CH_CUR_LN = .PUSH [PUSH$K_CUR_LIN];

```

```
3318 4774 2 BASSL_GOING_BACK = .PUSH [PUSH$L_GOING_BACK];
3319 4775 2 BASSA_RESTART = .PUSH [PUSH$L_RESTART];
3320 4776 2
3321 4777 2 !+ We are done with the item from error stack, free it.
3322 4778 2 !-
3323 4779 2 BEGIN
3324 4780 2
3325 4781 2 LOCAL
3326 4782 2 FREE_VM_RESULT;
3327 4783 2
3328 4784 2 FREE_VM_RESULT = LIB$FREE_VM (%REF (PUSH$L_LENGTH), PUSH);
3329 4785 2
3330 4786 2 IF ( NOT .FREE_VM_RESULT) THEN BASS$$STOP (BASS$L_PROLOSSOR);
3331 4787 2
3332 4788 2 END;
3333 4789 2 RETURN (SS$L_NORMAL);
3334 4790 1 END;
```

! of routine BASS\$POP_ERR

				00FC 00000	.ENTRY BASS\$POP_ERR, Save R2,R3,R4,R5,R6,R7	4717
		57	00000000	EF 9E 00002	MOVAB ERROR_STACK, R7	
		5E		08 C2 00009	SUBL2 #8, SP	
	04	AE	00	B7 0F 0000C	REMQUE @ERROR_STACK, PUSH	4757
				09 1C 00011	BVC 1\$	
		7E	00G	8F 9A 00013	MOVZBL #BASS\$L_PROLOSSOR, -(SP)	
	F49F	CF		01 FB 00017	CALLS #1, BASS\$\$STOP	
		56	04	AE D0 0001C	MOVL PUSH, R6	4763
		A7	08	A6 D0 00020	MOVL 8(R6), BASSL_ERRFLG	
D4	A7	OC	A6	08 28 00025	MOV3 #8, 12(R6), BASSL_ERRN	4764
	DC	A7	14	A6 7D 0002B	MOVQ 20(R6), BASSL_ERR	4765
	E8	A7	1C	A6 7D 00030	MOVQ 28(R6), HIGHEST_LEVEL	4767
	FO	A7	24	A6 7D 00035	MOVQ 36(R6), ACCUM_LEVEL	4769
	F8	A7	2C	A6 7D 0003A	MOVQ 44(R6), SYSTEM_ERROR	4771
	OC	A7	34	A6 7D 0003F	MOVQ 52(R6), BASSA_CH_CUR_LN	4773
	14	A7	3C	A6 D0 00044	MOVL 60(R6), BASSA_RESTART	4775
			04	AE 9F 00049	PUSHAB PUSH	4784
	04	AE	40	8F 9A 0004C	MOVZBL #64, 4(SP)	
			04	AE 9F 00051	PUSHAB 4(SP)	
	00000000G	00		02 FB 00054	CALLS #2, LIB\$FREE_VM	
		09		50 E8 0005B	BLBS FREE_VM_RESULT, 2\$	4786
		7E	00G	8F 9A 0005E	MOVZBL #BASS\$L_PROLOSSOR, -(SP)	
	F454	CF		01 FB 00062	CALLS #1, BASS\$\$STOP	
		50		01 D0 00067	MOVL #1, R0	4789
				04 0006A	RET	4790

; Routine Size: 107 bytes, Routine Base: _BASS\$CODE + 0D59

```
3335 4791 1
3336 4792 1 END
3337 4793 1
3338 4794 0 ELUDOM
```


BAS\$ERROR
1-074

I 14
16-Sep-1984 00:23:13
14-Sep-1984 11:54:56

VAX-11 BLISS-32 V4.0-742
[BASRTL.SRC]BASERROR.B32;1

Page 96
(30)

PSECT SUMMARY

Name	Bytes	Attributes
BAS\$DATA	68	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
BAS\$CODE	3524	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
_S255\$DUA28:[SYSLIB]STARLET.L32;1	9776	37	0	581	00:01.1

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/NOTRACE/LIS=LIS\$:BASERROR/OBJ=OBJ\$:BASERROR MSRC\$:BASERROR/UPDATE=(ENH\$:BASERROR)

: Size: 3012 code + 580 data bytes
: Run Time: 01:35.6
: Elapsed Time: 03:17.0
: Lines/CPU Min: 3009
: Lexemes/CPU-Min: 41958
: Memory Used: 550 pages
: Compilation Complete

0022 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

BASEDDFS
LIS

BASERROR
LIS

BASEDDDF
LIS

BASEDIT
LIS

BASEND
LIS

BASEDUP
LIS

BASEMULP
LIS

BASEDGSB
LIS

BASERTXT
LIS